Computer Science Honours

Literature Review

An Investigation of Digital Mixing and Panning Algorithms

Jessica Kent

Department of Computer Science, Rhodes University

Supervisor:
Richard Foss

Consultant:
Corinne Cooper

May 30, 2014

Abstract

The difference between the summing quality of analogue and digital audio is widely debated. Possible causes for this such as sampling rates, frequency response in human hearing and panning laws are considered. A study of how three open source Digital Audio Workstations (DAWs) define and implement gain, panning and summing of individual samples is then discussed. It was found that although samples were summed similarly, the execution of panning was distinct for each DAW. Previous testing of DAWs and alternate methods for summing samples were also reviewed.

Contents

1 Introduction

When audio files are added together, or summed, in a Digital Audio Workstation to create a final stereo track, the sound quality is missing the warmth and depth that the track would have if it had been summed using an analogue console. The hypothesis for this thesis is that a mechanism to provide the characteristics of an analogue mixer can be found and implemented in a digital mix.

2 Analogue vs Digital

In the sound engineering community there is debate about the sound quality of an audio mix that has been digitally summed instead of summed using an analogue mixing console. For the purpose of this paper, the terms "summing" and "mixing" will be used interchangeably and an "analogue" sample will refer to audio that has been summed in an analogue console, even if it played from a digital system. Some professionals say that a digital mix is lacking the "undeniable depth, width, punch and realism" (Farmelo, 2011, p. 1) of an analogue mix. However, as Leonard *et al.* (2012, p. 1) states, there is not much "quantifiable evidence to support these claims" yet. And although others, like Cochrane (2012), point out that summing with a DAW or analogue console won't necessarily improve the quality of a song if it has not been correctly mixed (artistically speaking), it is generally accepted that the limitations of analogue equipment might account for the difference in the perceived character of the mix (Cooper, 2004). For sound engineers who have been recording and mixing for the past 20 years, this unexplained variation of sound is a source of frustration, and summing boxes, like the Crane Song Egret or Rupert Neve Design 5059 (Sound on Sound, 2012), are being sold for the sole purpose of creating an analogue mix.

2.1 Definitions and Concepts in Audio Recording

In order to begin comparing digital and analogue mixing methods, some background information about audio recording is needed. Audio was first recorded using analogue equipment, usually onto magnetized tape. A recording device, typically a microphone, would record and convert fluctuations in air pressure to a measurable electronic signal. These varying electrical signals would then be recorded onto tape (Watkinson, 2001). As technology has advanced, digital equipment has been created to record, edit and mix audio signals. However,

microphones still capture analogue signals, so some kind of conversion needs to take place for audio to be processed using a Digital Audio Workstation (DAW). This has triggered many discussions about the quality variations between analogue and digital equipment and even among the "perceivable difference in sound quality between different DAWs" themselves, as stated by Leonard *et al.* (2012, p. 1).

To convert the recorded voltages to a digital audio signal, measurements of the voltages, called samples, are taken at fixed time intervals. This pulse code modulation (PCM) method is generally accepted as the standard system used in the conversion process (Lipshitz & Vanderkooy, 2004, p. 200). To make sure none of the original signal is lost, samples need to be taken at very high rates of 44.1kHz, 48kHz, 96kHz and 192kHz. An example of a wave being sampled at two different rates, the first one much lower than the other, is shown in Fig. 1. The rate of 44.1kHz was chosen to be used on digital compact discs (CDs) for numerous reasons - one of which is that it is more than twice the rate of the highest frequency humans can hear (Watkinson, 2001, pp. 207–209). This is a result of the Nyquist Theorem, which states that at least two sample points are needed to recreate a wave of a specific frequency (Lipshitz & Vanderkooy, 2004, p. 201). These samples are then stored numerically and can be used to recreate and modify the recorded sound wave (Watkinson, 2001).

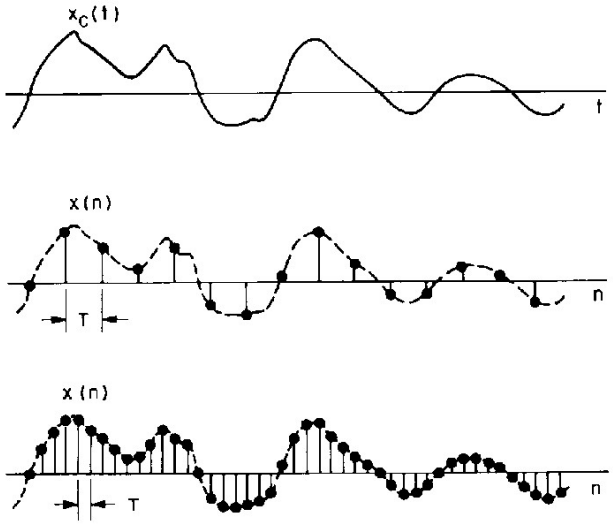

Figure 1: Sound wave sampled at two different rates (Rabiner, 1982, p. 80)

The amplitude or level of an audio wave is measured in deciBels - a logarithmic scale named after Alexander Graham Bell (Watkinson, 2001, p. 66).

The standard calculation used to convert a ratio value (generally between 0 and 1) to it's deciBel equivalent is given in Equation 1 (Watkinson, 2001, p. 66).

$$deciBels = 20 \cdot log_{10}(amplitude\_ratio) \tag{1}$$

Just as the maximum frequency that can be stored is determined by the choice of sample rate, the bit depth limits the amount of signal that can be stored by a digital system. When an input signal is too loud and the amplitude of the sound wave is too big for a digital system to store, the wave 'clips' causing the sound to distort (Cornell, 2009). The industry standard bit depth for CDs is 16-bit although audio is usually recorded and processed at 24-bit or 32-bit and converted to 16-bit after being mixed and mastered (Izhaki, 2008, p. 51).

2.2 Sound Waves and Sample Rates

A sound wave being produced by a musical instrument is typically a complex wave, made up of the fundamental frequency and higher harmonics (or overtones) that are a positive multiple of the fundamental frequency. In the 1800s, Joseph Fourier discovered that any existing sound wave can be represented mathematically as a sum of "simple harmonic terms" (Kinsler, 1962, p. 15), which are usually sine waves. This means that the sound wave in Fig. 2 can be expressed as the sum of two sine waves.

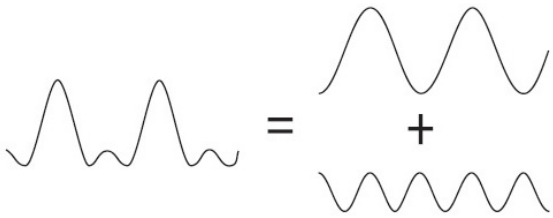

Figure 2: Sound wave comprised of two sine waves (Agilent Technologies, 2000, p. 7)

In conventional DAWs audio is recorded and summed at a sampling rate of either 44.1kHz or 48kHz. According to the Nyquist Theorem, frequencies above half the sampling rate (i.e. 22.05kHz or 24kHz respectively) are not recorded because there are not enough sample points to recreate them. However, as Izhaki (2008, pp. 456–457) points out, if a complex wave or any distortion is recorded at 48kHz, the harmonics of the wave above 24kHz will not

be recorded. This is not a problem on an analogue console. Although humans cannot hear sound waves higher than 20kHz (Watkinson, 2001, pp. 45-46), they are still recorded by the analogue console. When multiple audio tracks, one or more of which contains sound waves above 24kHz, are digitized and then summed together, it is theoretically possible that this high frequency wave will affect sounds waves below 20kHz. This would affect what humans can hear and could possibly be the difference heard when summing audio digitally.

However, Watkinson (2001, pp. 729-730) emphatically says that although demonstrations may have shown that higher sampling rates sound better, the experiments were not designed or carried out correctly. He also states that while converters sampling at 96kHz have been proven to sound better, "this does not prove that 96kHz is necessary" (Watkinson, 2001, p. 730) because a better designed converter sampling at a lower rate could produce the same results.

2.3 Fletcher-Munson curves

Not only are human ears limited to the frequencies they can hear, but they perceive some frequencies to be louder or softer than others. A set of experiments performed by Harvey Fletcher and W.A. Munson in 1933 (Izhaki, 2008, p. 12) resulted in the Feltcher-Munson curves (as shown in Fig. 3). These curves show that in order to hear a bass frequency under 100Hz as loud as a mid-range frequency of, say a 1kHz wave, the level of the audio needs to be higher. In other words, bass frequencies are not heard clearly in softer audio but our ears are very receptive to frequencies between about 2kHz and 5kHz (Watkinson, 2001, p. 45).

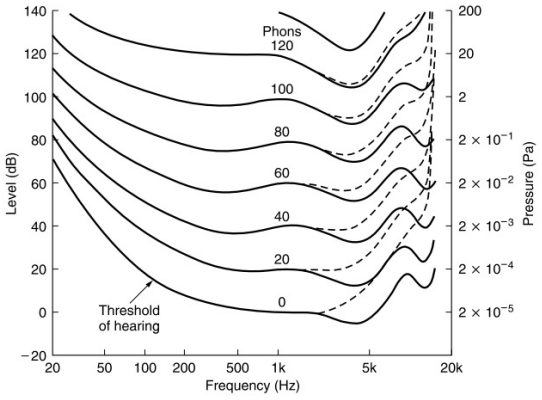

Figure 3: Fletcher-Munson curves (Watkinson, 2001, p. 46)

2.4 Panning Laws

When creating a stereo mix of a song, individual tracks are usually panned to change the horizontal location that they appear to originate from. The most common way to compute this position in a DAW is to use a "sine-cosine" calculation or panning law (Griesinger, 2002, p. 1). A single track is panned in a DAW using some sort of digital panning control. The specified panning is then converted to an angle between 0 and 90 degrees (as shown in Fig. 4) representing a panning anywhere from hard left to hard right respectively (Griesinger, 2002). The output of the stereo track is calculated using Equations 2 and 3, where a is the angle. Although these equations provide constant loudness, where $left^2 + right^2 = input^2$ (Griesinger, 2002), when tracks are center-panned there is a 3dB level increase heard to be coming from the center. To compensate for this, a panning law can be used.

$$left = cos(a) \cdot input \qquad (2) \qquad right = sin(a) \cdot input \qquad (3)$$

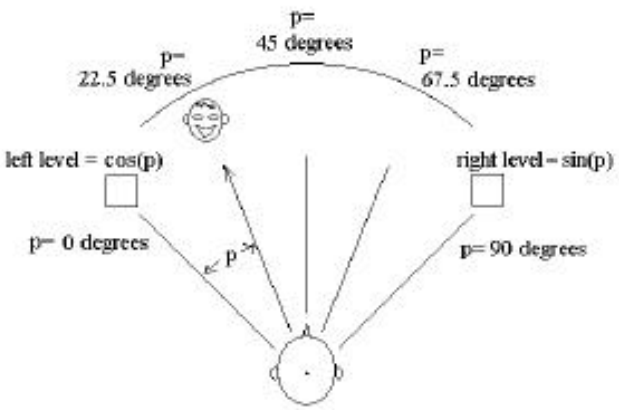

Figure 4: Angle of panning (Griesinger, 2002)

The four most frequently used panning laws are the 0dB, -3dB, -4.5dB and -6dB panning laws (Izhaki, 2008, pp. 189-193) where the values refer to the total reduction of deciBels on center-panned tracks. It is well-established that the "-3dB pan law is generally the best option when stereo mixing" Izhaki (2008, p. 193) but not all DAWs use this as the default panning law. For example, Rosegarden uses the 0dB as the default panning law (see AudioLevel.cpp in Appendix C).

2.5 Analogue Equipment

When examining various digital summing algorithms it is useful to investigate how an analogue mixer combines signals to produce a summed output. An example of a summing amplifier circuit is shown in Fig. 5 where three input signals (V_1, V_2 and V_3) are added together to produce an output voltage ($-V_{out}$). When the input impedances ($-R_{IN}$) are the same, the output can be calculated using Equation 4. This means that the output voltage is proportional to the sum of the input voltages (Storr, 2014).

$$-V_{out} = \frac{R_F}{R_{In}}(V_1 + V_2 + V_3) \tag{4}$$

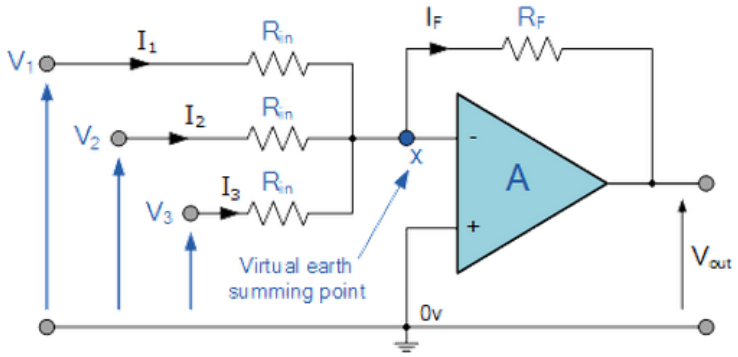

Figure 5: Summing Amplifier Circuit (Storr, 2014)

3 Mixing Algorithms

Since a song typically consists of multiple tracks, a DAW needs to be able to sum these separate tracks to create a master stereo track. This process is not always as simple as adding the two values together, as this new summed value could clip or "over- or underflow the range available" as Vogler (2012, p. 1) puts it. Therefore various mixing algorithms have been developed which aim to add the tracks together in such a way that the level of each individual track is not perceived to be louder or softer than it was originally. There is a diverse selection of DAWs available and industry recording engineers argue about which DAW is technically the best for summing audio. For the purposes of this investigation, the three DAWs that were selected are all free, open source applications. These applications are Audacity (2014b), Ardour (2014) and Rosegarden (2014). They were chosen with the aim of examining the source

code to determine the specific summing algorithms, gain representation and panning laws used, as these are the three properties that could be responsible for the quality difference.

3.1 Representation of Gain

Audacity's interface allows the user to choose between a linear or logarithmic scale to represent the gain on the meter display. The linear scale has a minimum of 0 and maximum of 1 while the default logarithmic scale has a maximum level of 0dB (Audacity, 2014a). Any audio that exceeds these maximum values is shown to be clipping. The methods *ToDB* and *ToLinearIfDB* (see Meter.cpp in Appendix A), based on Equation 1, are used to convert the user's chosen gain between representations. However, all sample calculations use the linear scale and gain is represented as a double floating point value between 0 and 1.

Ardour makes use of floating point representation to store the gain value, which can be between $-\infty$ and 6dB using the logarithmic scale and a floating point value between 0 and 2 using the linear scale (Ardour, 2014). The conversion calculations used are also based on Equation 1 and can be seen in dB.h in Appendix B.

Rosegarden provides five distinct fader types (see AudioLevel.cpp in Appendix C), each with specific minimum, maximum and step values. Depending on the settings, the user can choose a gain value between -70dB and 10dB. This value is converted to a floating point value between 0 and 10 using the *dB_to_multiplier* method in Appendix C. Interestingly, Rosegarden uses a base factor of 10 instead of using 20 as Audacity and Ardour do.

3.2 Panning Laws

Audacity's panning control allows the user to specify a value between -1 and 1 in increments of 0.1 (see ASlider.cpp in Appendix A) which is displayed as being a percentage left or right as depicted in Fig. 6. The method *GetChannelGain* (see WaveTrack.cpp in Appendix A) calculates the gain for either a left or right output by using Equation 5 to get a positive pan value and multiplying that by the user-specified gain to find the gain for the channel. This is not the sin-cos panning law discussed earlier and Audacity therefore does not account for an increase in level when tracks are center-panned.

$$\begin{cases} right = pan + 1.0 & pan < 0 \\ left = 1.0 - pan & pan > 0 \end{cases} \tag{5}$$

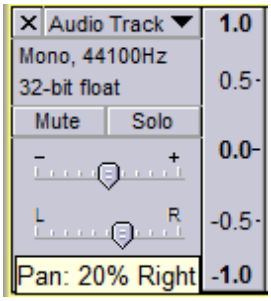

Figure 6: Panning slider set at "20% Right" in Audacity (Audacity, 2014b)

Ardour defines their panning in terms of Azimuth angles, a coordinate system using North and East as reference points (Peat, n.d.). The user can set the panning anywhere between 0 and 180 degrees (Ardour, 2014) and this is converted to a double floating point value between 0 and 1 using the method *azimuth_to_lr_fract* (see panner.h in Appendix B). In the method *distribute_one* (from panner_2in2out.cc in Appendix B), Ardour applies the panning to the audio samples.

In the Rosegarden code AudioLevel.cpp (see Appendix C), the various panning laws are given in the panGainRight method, where a pan value between -100 and 100 is converted to a value between 0 and 2 (Rosegarden, 2014). The -0db and -6db laws add 100 to the panning value and then divide this sum by either 100 or 200 while both -3dB panning laws return the square root of this calculation. The overall gain and panning for the left and right channel is applied to the samples in the *setBussLevels* method in the Rosegarden code AudioProcess.cpp.

It is interesting to note that each DAW defined and handled panning uniquely. This may be one of the reasons why digitally summed audio is difficult to test and compare, as remarked by Leonard *et al.* (2012) in their investigations of DAW summing.

3.3 Summing of Samples

In Audacity the samples, after being multiplied by the selected gain, are simply added together and it is up to the user to avoid clipping. As can be seen in Mix.cpp in Appendix A the summing process takes place in the *MixBuffers method*, which loops through all the audio samples and multiplies each one by the gain of the channel (a floating value between 0 and 1) before adding it to the destination buffer.

Ardour implements the mixing process in the appropriately named mix.cc (see Appendix B) where two methods for mixing the samples are given - one

including gain and one without. These methods resemble the implementation by Audacity: the samples are added together and stored in the destination buffer. The *default_mix_buffers_with_gain* method multiplies the samples by a floating point gain value (between 0 and 2, as previously mentioned) before adding them.

Rosegarden first adds the samples from each channel together and then adds them to a destination buffer (see PlayableAudioFile.cpp in Appendix C) after having applied panning and gain in the *setBussLevels* method.

Although different gain and pan algorithms are used, each DAW sums the samples in very similar ways. This means there is an opportunity to modify and experiment with this process when implementing new mixing algorithms to emulate the analogue mixing process.

3.4 Alternative Summing Methods

Since the effect of audio summing methods has been so widely debated, alternative methods for adding the samples together have been developed. For example, Viktor T Toth devised Equation 6 to calculate the summed value of two samples a and b that themselves have values between 0 and 1. This formula can be extended for three samples as in Equation 7. He refined the formula to account for samples that could be added together without clipping to get Equation 8. However, as Vogler (2012) points out, this method is not symmetrical (see Figure 7) and favours extreme values in the subsequent mix.

$$sum = a + b - ab \tag{6}$$

$$sum = a + b + c - ab - ac - bc + abc \tag{7}$$

$$sum = \begin{cases} 2ab & a < 0.5 \cap b < 0.5 \\ 2(a+b) - 2ab - 1 & a \geq 0.5 \cup b \geq 0.5 \end{cases} \tag{8}$$

Vogler (2012) himself developed a method for adding two samples together which he calls "Loudness Normalization by Logarithmic Dynamic Range Compression". He begins by stating that the easiest method for making sure two samples stay within the output range is to divide by 2 as in Equation 9. However, if one sample was very quiet the other sample would effectively be half as loud as it was originally. He decided to then dynamically compress the output level only if it was above a specific threshold. This would ensure that most

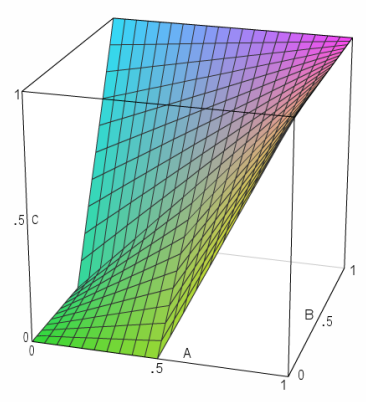

Figure 7: Viktor T. Toth method (Vogler, 2012)

samples could be added together without further calculations. This Linear Dynamic Range Compression method is presented in Equation 10 where $x = a+b$ and is a floating point value between 0 and 2 and t is the chosen threshold between 0 and 1. As can be seen in Fig. 8, where a threshold of 0.6 is used, the compression takes effect pretty dramatically, which results in an audible reduction in level. To lesson the impact of this, Vogler (2012) used mathematics beyond the scope of this paper to produce a Logarithmic Dynamic Range Compression method given in Equation 11. This results in a smoother transition as compression is applied as shown in Fig. 9. It would be interesting to see what effect implementing these alternative summing methods would have on the sound quality of a digital mix.

$$sum = \frac{a+b}{2} \tag{9}$$

$$sum = \begin{cases} x & -t \leq x \leq t \\ \frac{x}{|x|} \cdot (t + \frac{1-t}{2-t} \cdot (|x| - t) & |x| > t \end{cases} \tag{10}$$

$$sum = \begin{cases} x & -t \leq x \leq t \\ \frac{x}{|x|} \cdot (t + (1-t) \cdot \frac{ln(1+f_\alpha(t) \cdot \frac{|x|-t}{2-t})}{ln(1+f_\alpha(t))} \cdot & |x| > t \end{cases} \tag{11}$$

4 Testing

After implementing these panning and mixing algorithms, some sort of testing should be conducted in order to make any conclusions about their effect on

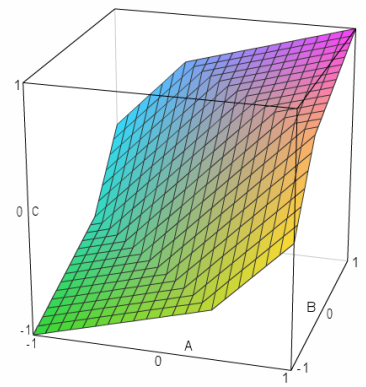

Figure 8: Linear Dynamic Range Compression (t = 0.6) (Vogler, 2012)

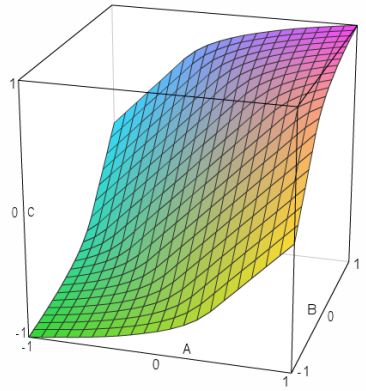

Figure 9: Logarithmic Dynamic Range Compression (t = 0.6) (Vogler, 2012)

the overall sound quality of the mix. As far as audio testing is concerned, both objective and subjective testing is normally performed as music has both technical and artistic properties. As Watkinson (2001, p. 708) explains, combining both approaches is "the only way to achieve outstanding results".

4.1 Objective Testing Methods

In experiments with investigations similar to this one, objective testing has usually been performed using visual testing and difference methods (Leonard *et al.*, 2012). Objective testing includes measuring the difference between a sound wave and the same sound wave after it has been processed in some way. Results should be reproducible in order for the test to be seen as a valid method for analyzing a particular system (Watkinson, 2001, p. 710). However, conclusions drawn from objective tests do not necessarily indicate how well a system will perform in the real world. This is why subjective testing is usually employed in combination with objective testing. Visual or difference testing as used in the experiments by Leonard *et al.* (2012) will be used to examine if any differences exist between the summing of analogue and digital sound waves in the subsequent thesis.

4.2 Subjective Testing Methods

For subjective testing to be accurate, the loudspeakers being used need to be as accurate as possible (Watkinson, 2001, p. 720). Listening tests are the most common method used to subjectively compare audio samples but it is difficult to remove bias from the results (Watkinson, 2001, p. 709). In the experiments conducted by Leonard *et al.* (2012) listening tests were carried out on three specifically chosen DAWs out of the five that were originally analyzed. When conducting digital listening tests the interface used to switch between and compare samples is very important and this needs to eliminate any possible bias.

4.3 Previous Testing

There have been very few official tests done as far as analogue or digital summing is concerned. The most recent testing was conducted by Leonard *et al.* (2012) where the internal summing of five different DAWs was investigated. They decided that they needed to test three specific aspects of each DAW to find out which of the factors, if any, were the source of the differences. They chose to examine how the input was read in and any error correction done, the

internal summing of the sources and how the gain was changed in each DAW. After some initial testing (Leonard & Buttner-Schnirer, 2012) they discovered an error in the panning of one of the tracks. This resulted in an examination of the effect that panning had on the summing of the tracks. They found that the different panning laws of the DAWs caused significant variations in output levels and sound quality. They then investigated the panning laws of each DAW and found the panning technique to be inconsistent across DAWs, especially when center-panning the audio tracks instead of hard panning left or right (Leonard *et al.*, 2012). After retesting and normalizing the audio, they discovered that there was no difference in the addition of the samples between any of the DAWs. They did, however, find an objective and subjective difference in level on the summing of tracks that were center-panned even when the same panning law was used across all DAWs. They suggest that the most problematic component appears to be the individual panning laws utilized by each DAW and recommend further testing of DAW panning laws to explore the effect on sound quality.

Another experiment done by Aarts *et al.* (2007) compared two DAWs with different sampling rates to an analogue system in order to find the most similar sounding DAW. Their tests showed that the DAW with the higher sampling rate was the system that sounded most like the analogue system. They do not, however, say if there were any audible differences between the analogue and digital systems.

4.4 Timing

If audio is to be mixed and played back in real time, the summing algorithm needs to be extremely efficient to calculate the sum as fast as possible. When a sample rate of 48kHz is chosen the instructions used to add the samples need to be executed in under $\frac{1}{48000}$ of a second or 20.83 μs. If a higher sample rate is used then the summing needs to be performed even faster.

5 Summary

Overall, it appears that in the process of summing audio samples in a digital workstation, there are various elements that could influence the sound quality of a mix. It is possible that an analogue summing algorithm could be created by operating at higher sample rates, applying alternative mixing methods when summing samples or modifying panning laws. Once various emulation algorithms have been created, they need to be tested both objectively and

subjectively to find the sound quality most similar to an analogue mixer. If these algorithms are to be played back in real time they need to be timed to ensure they are quick and efficient enough for high sample rates.

References

Aarts, Ronald, Engel, Jan, Reefman, Derk, Usher, John, & Woszczyk, Wieslaw. 2007 (Jun). Which of the Two Digital Audio Systems Best Matches the Quality of the Analog System? *In: Audio Engineering Society Conference: 31st International Conference: New Directions in High Resolution Audio.*

Agilent Technologies. 2000. The Fundamentals of Signal Analysis: Application Note 243. Online. Available from: `http://cp.literature.agilent.com/litweb/pdf/5952-8898E.pdf`. Accessed on: 20 May 2014.

Ardour. 2014. Ardour Website. Online. Available from: `http://ardour.org/`. Accessed on: 25 May 2014.

Audacity. 2014a. Audacity Meter Toolbar. Online. Available from: `http://manual.audacityteam.org/o/man/meter_toolbar.html`. Accessed on: 26 May 2014.

Audacity. 2014b. Audacity Website. Online. Available from: `http://audacity.sourceforge.net/`. Accessed on: 25 May 2014.

Cochrane, Graham. 2012. Analog Summing And Why You Shouldnt Care. Self-published. Available from: `http://therecordingrevolution.com/2012/05/04/analog-summing-and-why-you-shouldnt-care/`. Accessed on: 20 May 2014.

Cooper, Paul. 2004. Q. Is analogue mixing superior to digital summing? Sound on Sound. Available from: `http://www.soundonsound.com/sos/jun04/articles/qa0604-5.htm`. Accessed on: 28 February 2014.

Cornell. 2009. Bit Depth. Online. Available from: `http://digital.music.cornell.edu/cemc/book/export/html/1594`. Accessed on: 29 May 2014.

Farmelo, Allen. 2011. Analog vs. Digital Summing. Self-published. Available from: `http://www.farmelorecording.com/in-the-press/analog-vs-digital-summing/`. Accessed on: 28 February 2014.

Griesinger, David. 2002 (Apr). Stereo and Surround Panning in Practice. *In: Audio Engineering Society Convention 112.*

Izhaki, R. 2008. *Mixing Audio: Concepts, Practices and Tools.* Electronics & Electrical. Focal Press.

Kinsler, L.E. 1962. *Fundamentals of Acoustics.* 2nd edn. Wiley.

Leonard, Brett, & Buttner-Schnirer, Padraig. 2012 (Apr). Subjective Differences in Digital Audio Workstation Math. *In: Audio Engineering Society Convention 132*.

Leonard, Brett, Levine, Scott, & Buttner-Schnirer, Padraig. 2012 (Oct). Objective and Subjective Evaluations of Digital Audio Workstation Summing. *In: Audio Engineering Society Convention 133*.

Lipshitz, Stanley P., & Vanderkooy, John. 2004. Pulse-Code Modulation–An Overview. *J. Audio Eng. Soc*, **52**(3), 200–215.

Peat, Chris. Online. Available from: `http://www.heavens-above.com/glossary.aspx?term=azimuth`. Accessed on: 30 May 2014.

Rabiner, L. R. 1982 (Jun). Digital Techniques for Changing the Sampling Rate of a Signal. *In: Audio Engineering Society Conference: 1st International Conference: Digital Audio*.

Rosegarden. 2014. Rosegarden Website. Online. Available from: `http://www.rosegardenmusic.com/`. Accesed on: 25 May 2014.

Sound on Sound. 2012. Analogue Summing Mixers. Sound on Sound. Available from: `http://www.soundonsound.com/sos/jun12/articles/spotlight-0612.htm`. Accessed on: 20 May 2014.

Storr, Wayne. 2014. The Summing Amplifier. Online. Available from: `http://www.electronics-tutorials.ws/opamp/opamp_4.html`. Accessed on: 28 May 2014.

Vogler, Paul. 2012. Mixing two digital audio streams with on the fly Loudness Normalization by Logarithmic Dynamic Range Compression. Self-published. Available from: `http://voegler.eu/pub/audio/digital-audio-mixing-and-normalization.html#idx_3`. Accessed on: 15 February 2014.

Watkinson, J. 2001. *The art of digital audio*. Third edn. Focal Press.

A Audacity Code

Gain Representation from Meter.cpp

```
...
static float ToDB(float v, float range)
{
   double db;
   if (v > 0)
      db = 20 * log10(fabs(v));
   else
      db = -999;
   return ClipZeroToOne((db + range) / range);
}

double Meter::ToLinearIfDB(double value)
{
   if (mDB)
      value = pow(10.0, (-(1.0-value)*mDBRange)/20.0);
   return value;
}
```

Pan definition from ASlider.cpp

```
...
case PAN_SLIDER:
      minValue = -1.0f;
      maxValue = +1.0f;
      stepValue = 0.1f;
      orientation = wxHORIZONTAL;
      break;
...
```

Pan Algorithm from WaveTrack.cpp

```
float WaveTrack::GetChannelGain(int channel)
{
   float left = 1.0;
   float right = 1.0;
         ...
   if (mPan < 0)
      right = (mPan + 1.0);
   else if (mPan > 0)
```

```
        left = 1.0 - mPan;
    if ((channel%2) == 0)
        return left*mGain;
    else
        return right*mGain;
```

Summing Algorithm from Mix.cpp

```
void MixBuffers(int numChannels, int *channelFlags, float *gains,
                samplePtr src, samplePtr *dests,
                int len, bool interleaved)
{
        ...
        float gain = gains[c];
        float *dest = (float *)destPtr;
        float *temp = (float *)src;
        for (int j = 0; j < len; j++) {
            *dest += temp[j] * gain;    // the actual mixing process
            dest += skip;
        }
    }
```

B Ardour Code

Gain Representation from dB.h

```
static inline float dB_to_coefficient (float dB) {
        return dB > -318.8f ? pow(10.0f, dB * 0.05f) : 0.0f;
}

static inline float accurate_coefficient_to_dB (float coeff) {
        return 20.0f * log10f (coeff);
}
```

Panner conversion from panner.h

```
static double azimuth_to_lr_fract (double azi) {
        /* 180.0 degrees=> left => 0.0 */
        /* 0.0 degrees => right => 1.0 */
        return 1.0 - (rint(azi)/180.0);
}
static double lr_fract_to_azimuth (double fract) {
```

```
        /* fract = 0.0 => degrees = 180.0 => left */
        /* fract = 1.0 => degrees = 0.0 => right */
        return rint (180.0 - (fract * 180.0));
}
```

Panning Algorithm from panner_2in2out.cc
Note: A very similar method can be found in panner_1in2out.cc

```
void
Panner2in2out::distribute_one (AudioBuffer& srcbuf,
BufferSet& obufs, gain_t gain_coeff,
pframes_t nframes, uint32_t which)
{
...
/* LEFT OUTPUT */
...
for (n = 0; n < limit; n++) {
        left_interp[which] = left_interp[which] + delta;
        left[which] = left_interp[which] +
        0.9 * (left[which] - left_interp[which]);
        dst[n] += src[n] * left[which] * gain_coeff;
}
pan = left[which] * gain_coeff;

mix_buffers_with_gain (dst+n,src+n,nframes-n,pan);
...
/* RIGHT OUTPUT */
...
for (n = 0; n < limit; n++) {
        right_interp[which] = right_interp[which] + delta;
        right[which] = right_interp[which] +
        0.9 * (right[which] - right_interp[which]);
        dst[n] += src[n] * right[which] * gain_coeff;
}
pan = right[which] * gain_coeff;
mix_buffers_with_gain(dst+n,src+n,nframes-n,pan);
}
```

Mixing Algorithm from mix.cc

```
...
```

```
void default_mix_buffers_with_gain (ARDOUR::Sample * dst,
                const ARDOUR::Sample * src,
                pframes_t nframes, float gain)
{
        for (pframes_t i = 0; i < nframes; i++) {
                dst[i] += src[i] * gain;
        }
}
void default_mix_buffers_no_gain (ARDOUR::Sample * dst,
                const ARDOUR::Sample * src, pframes_t nframes)
{
        for (pframes_t i=0; i < nframes; i++) {
                dst[i] += src[i];
        }
}
```

C Rosegarden Code

Gain Representation from AudioLevel.cpp

```
...
const float AudioLevel::DB_FLOOR = -1000.0;
...
static const FaderDescription faderTypes[] = {
    FaderDescription(-40.0,  +6.0, 0.75), // short
    FaderDescription(-70.0, +10.0, 0.80), // long
    FaderDescription(-70.0,   0.0, 1.00), // IEC268
    FaderDescription(-70.0, +10.0, 0.80), // IEC268 long
    FaderDescription(-40.0,   0.0, 1.00), // preview
};
...
float
AudioLevel::multiplier_to_dB(float multiplier)
{
    if (multiplier == 0.0) return DB_FLOOR;
    float dB = 10 * log10f(multiplier);
    return dB;
}
```

```
float
AudioLevel::dB_to_multiplier(float dB)
{
    if (dB == DB_FLOOR) return 0.0;
    float m = powf(10.0, dB / 10.0);
    return m;
}
```

Panning Laws and Conversion from AudioLevel.cpp

```
int AudioLevel::m_panLaw = 0;
...
float
AudioLevel::panGainRight(float pan)
// Apply panning law to right channel
{
    if (m_panLaw == 3) {
                // -3dB pannig law (variant)
        return sqrtf(fabsf((100.0 + pan) / 100.0));

    } else if (m_panLaw == 2) {
                // -6dB pan law
        return (100.0 + pan) / 200.0;

    } else if (m_panLaw == 1) {
                 // -3dB panning law
        return sqrtf(fabsf((100.0 + pan) / 200.0));

    } else {
                // 0dB panning law (default)
        return (pan < 0.0) ? (100.0 + pan) / 100.0 : 1.0;

    }
}
```

Applying Gain and Panning to Samples from AudioProcess.cpp

```
float gain[2];
gain[0] = rec.gainLeft;
gain[1] = rec.gainRight;
void AudioBussMixer::setBussLevels(int bussId,
                float dB, float pan)
```

```
{
        ...
    float volume = AudioLevel::dB_to_multiplier(dB);
    rec.gainLeft = volume *
        ((pan > 0.0) ? (1.0 - (pan / 100.0)) : 1.0);
    rec.gainRight = volume *
        ((pan < 0.0) ? ((pan + 100.0) / 100.0) : 1.0);
        ...
}
```

Mixing Process from PlayableAudioFile.cpp

```
for (size_t i = 0; i < n; ++i) {
        sample_t v = cached[0][scanFrame + i]
+ cached[1][scanFrame + i];
        destination[0][i + offset] += v;
}
```