AALTO UNIVERSITY SCHOOL OF SCIENCE AND TECHNOLOGY
Faculty of Electronics, Communication and Automation
Department of Signal Processing and Acoustics

Antti Huovilainen

Design of a Scalable Polyphony-MIDI Synthesizer for a Low Cost DSP

Master's Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology.

Espoo, May 25, 2010

Supervisor: Professor Vesa Välimäki
Instructors: Professor Vesa Välimäki

AALTO UNIVERSITY
SCHOOL OF SCIENCE AND TECHNOLOGY

ABSTRACT OF THE
MASTER'S THESIS

Author:	Antti Huovilainen

Name of the thesis:	Design of a Scalable Polyphony-MIDI Synthesizer for a Low Cost DSP
Date:	May 25, 2010 **Number of pages:** 55+x
Faculty:	Electronics, Communication and Automation
Professorship:	Audio signal processing (S-89)
Supervisor:	Prof. Vesa Välimäki
Instructors:	Prof. Vesa Välimäki

In this thesis, the design of a music synthesizer implementing the Scalable Polyphony-MIDI soundset on a low cost DSP system is presented. First, the SP-MIDI standard and the target DSP platform are presented followed by review of commonly used synthesis techniques and their applicability to systems with limited computational and memory resources.

Next, various oscillator and filter algorithms used in digital subtractive synthesis are reviewed in detail. Special attention is given to the aliasing problem caused by discontinuities in classical waveforms, such as sawtooth and pulse waves and existing methods for bandlimited waveform synthesis are presented.

This is followed by review of established structures for computationally efficient time-varying filters. A novel digital structure is presented that decouples the cutoff and resonance controls. The new structure is based on the analog Korg MS-20 lowpass filter and is computationally very efficient and well suited for implementation on low bitdepth architectures.

Finally, implementation issues are discussed with emphasis on the Differentiated Parabole Wave oscillator and MS-20 filter structures and the effects of limited computational capability and low bitdepth. This is followed by designs for several example instruments.

Keywords: sound synthesis, computer music, computational efficiency, synthesis algorithms, sound design.

AALTO-YLIOPISTON
TEKNILLINEN KORKEAKOULU

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Antti Huovilainen
Työn nimi:	Design of a Scalable Polyphony-MIDI Synthesizer for a Low Cost DSP
Päivämäärä:	25. toukokuuta 2010 **Sivuja:** 55+x
Tiedekunta:	Elektroniikka, tietoliikenne ja automaatio
Professuuri:	Audiosignaalinkäsittely (S-89)
Työn valvoja:	Prof. Vesa Välimäki
Työn ohjaajat:	Prof. Vesa Välimäki

Tässä diplomityössä esitetään Scalable Polyphony-MIDI-standardin soitinvalikoiman toteuttavan musiikkisyntetisaattorin suunnittelu edulliselle signaaliprosessorille. Ensiksi esitellään SP-MIDI-standardi ja käytettävä signaaliprosessori. Sen jälkeen kerrataan yleisesti käytössä olevia synteesitekniikoita, ja niiden soveltuvuutta järjestelmiin, joissa laskentateho ja muistin määrä ovat rajoittuneita.

Seuraavaksi käydään yksityiskohtaisesti läpi vähentävässä synteesissä käytettäviä oskillaattori- ja suodintekniikoita. Erityistä huomiota kiinnitetään laskostumiseen, joka johtuu perinteisissä aaltomuodoissa, kuten sahalaita- ja pulssi-aallossa, olevista epäjatkuvuuksista, ja olemassaolevia kaistarajoitettuja aaltomuotosynteesimenetelmiä kerrataan.

Tämän jälkeen kerrataan olemassaolevia rakenteita laskennallisesti tehokkaille aikavarianteille suotimille. Kokonaan uusi, rajataajuuden ja resonanssin ohjauksen erottava, suodinrakenne esitellään. Analogiseen Korg MS-20-alipäästösuotimeen perustuva rakenne on laskennallisesti erittäin tehokas ja soveltuu hyvin toteutettavaksi vähäbittisillä arkkitehtuureilla.

Lopuksi käsitellään toteutukseen liittyviä yksityiskohtia kiinnittäen erityistä huomiota differentioituun paraabeliaaltoon ja MS-20-suotimeen ja rajoitetun laskentakapasiteetin ja laskentaresoluution vaikutuksiin. Tämän jälkeen esitetään joidenkin esimerkkisoitinten toteutus.

Avainsanat: äänisynteesi, tietokonemusiikki, laskennallinen tehokkuus, synteesialgoritmit, äänisuunnittelu.

Acknowledgements

This Master's thesis has been done in the Laboratory of Acoustics and Audio Signal Processing at Aalto University School of Science and Technology. The research was funded by VLSI Solution Oy, and was conducted during years 2004-2006.

I want to thank Professor Vesa Välimäki for his guidance and support and for his very considerable patience when the finishing of this thesis was delayed. I would also like to thank Dr. Teppo Karema and Mr. Tomi Valkonen for comments and discussion.

My gratitude also goes to the personnel of Acoustics Laboratory for many fruitful and interesting discussions both related and unrelated to audio processing.

Finally, I would like to thank people on EFNet #musicdsp for many useful ideas. Andrew Simper deserves special thanks for reviewing the thesis and giving last minute advice.

Otaniemi, May 25, 2010

Antti Huovilainen

Contents

Abbreviations

ADPCM	Adaptive Delta Pulse Code Modulation
ADSR	Attack-Decay-Sustain-Release
BLAMP	Bandlimited Ramp
BLEP	Bandlimited Step
BLIT	Bandlimited Impulse Train
CPU	Central Processing Unit
DPW	Differentiated Parabole Wave
DSP	Digital Signal Processor
FIR	Finite Impulse Response
FM	Frequency Modulation
IIR	Infinite Impulse Response
LFO	Low Frequency Oscillator
LSB	Least Significant Bit
MIDI	Musical Instrument Digital Interface
RAM	Random-access Memory
ROM	Read-only Memory
SP-MIDI	Scalable Polyphony-MIDI
SVF	State Variable Filter

List of Figures

List of Tables

Chapter 1

Introduction

Advances in computing power, miniaturization and communication bandwidth have revolutionized personal music listening in the last decade. It is now common for people to have multiple personal audio devices that are small enough that they can be always carried with the person. Although MP3 and other digital audio formats are very popular, there is often a need to provide music that requires much less storage and transfer bandwidth. Typical examples are polyphonic ringtones for mobile phones, background music in games and user provided musical content. Scalable Polyphony Musical Instrument Digital Interface (SP-MIDI) [28] is a recent standard for providing music in a format that requires little storage and is suitable for devices with limited computational power.

This thesis presents techniques that are suitable for synthesizing SP- MIDI music on such devices. First, an overview of the SP-MIDI standard and the target platform are given. Chapter 2 then gives an overview of the relevant synthesis techniques. Chapters 3 and 4 provide more detailed review of methods for digital oscillators and filters as used in some of the synthesis techniques. Chapter 5 describes specific implementation choices and issues, and chapter 6 discusses sound design and provides details of several example instruments. Finally chapter 7 concludes this thesis.

1.1 Scalable Polyphony-MIDI

Musical Instrument Digital Interface (MIDI) [28] is a standard that provides a way to transfer and store musical information, without containing actual audio data.

Scalable Polyphony-MIDI [29] is a standard by the MIDI Manufacturers Association. The main extension over the General MIDI standard is that it specifies ways to limit the polyphony as necessary. This is a common need in mobile and other multimedia appliances where the computing power is limited or where the appliance performs multiple tasks

simultaneously.

SP-MIDI and General MIDI music is normally provided in Standard MIDI File format [28]. They contain the song data as MIDI messages, which can be note on/off, program change, pitch bend and other performance data. In addition to this, they may contain metadata such as tempo changes. SP-MIDI defines a particularly important set of metadata, which specifies how notes are to be dropped when the maximum polyphony of the synthesizer is exceeded. MIDI files do not typically contain any actual sound data, although there are extensions that provide for this [30].

The SP-MIDI standard defines a minimum set of supported instruments. Table 1.1 lists the required 13 melodic instruments. These contain one instrument from each major group of instruments so that unsupported instruments are mapped to the main instrument in the group. The table also lists the range (in MIDI note numbers and Hz) that each instrument should support as a minimum.

MIDI note numbers are defined such that note 60 corresponds to the middle C note of piano and there are 12 notes in an octave. The equation for frequency f of note k is therefore

$$f = f_0 2^{(k-69)/12} \tag{1.1}$$

where f_0 is the frequency of middle A, defined to be 440 Hz for standard tuning.

SP-MIDI also specifies 13 percussion instruments listed in table 1.1 and 9 sound effects listed in table 1.1. The percussion instruments use a similar mapping technique where unsupported instruments are mapped to the closest supported instruments.

The specification does not say how the instrument sounds should be produced or list a set of required attributes from the sound apart of the instrument name and range. Thus a good method to gauge what a particular instrument should sound like is to listen to existing General MIDI implementations in commercial synthesizers. This is also prudent as the MIDI files to be played back are likely produced on a different system from the playback system.

1.2 Low cost digital signal processor

The design was to be implemented in the (then upcoming) VLSI Solution VS1003 MP3 playback IC [52]. VS1003 is a single chip solution for playback of MP3, WMA or ADPCM encoded sound files and SP-MIDI music files. It contains a DSP core, embedded memory for the decoding of sound stream, 16-bit stereo digital to analog converter and an integrated headphone amplifier. As the topic of the work was design only, exact knowledge of the IC was not required. General specifications were still given to ensure the algorithms and instruments could later be efficiently implemented on the target platform.

MIDI program	Instrument name	Range (MIDI notes)	Range (Hz)
1	Acoustic grand piano	21 - 108	27.50 - 4186
12	Vibraphone	53 - 89	176.4 - 1397
17	Drawbar organ	46 - 96	116.5 - 2093
28	Electric guitar (clean)	40 - 86	82.41 - 1175
34	Electric bass (finger)	28 - 55	41.20 - 196.0
41	Violin	55 - 96	196.0 - 2093
49	String Ensemble 1	28 - 96	41.20 - 2093
57	Trumpet	58 - 94	233.1 - 1865
67	Tenor sax	42 - 75	92.50 - 622.3
74	Flute	60 - 96	261.5 - 2093
82	Lead 2 (sawtooth)	21 - 108	27.00 - 4186
90	Pad 2 (warm)	36 - 96	36.41 - 2093
115	Steel drums	52 - 76	164.8 - 659.3

Table 1.1: List of melodic instruments in SP-MIDI

MIDI note number	Instrument name
36	Bass drum 1
40	Electric snare
42	Closed hi-hat
45	Low tom
46	Open hi-hat
49	Crash cymbal 1
50	High tom
51	Ride cymbal 1
54	Tambourine
63	Mute high conga
64	Low conga
70	Maracas
75	Claves

Table 1.2: List of percussion instruments in SP-MIDI

MIDI program	Instrument name
120	Reverse cymbal
121	Guitar fret noise
122	Breath noise
123	Seashore
124	Bird tweet
125	Telephone ring
126	Helicopter
127	Applause
128	Gunshot

Table 1.3: List of sound effect instruments in SP-MIDI

The VS1003 core is similar to other 16-bit DSPs. It has a 16-bit multiplier with a 40-bit accumulator and can perform one multiply and accumulate operation and two data accesses per instruction cycle. The available resources were estimated to be

- 36 MHz instruction rate
- 20 - 30 kilobytes of sample ROM
- 1 - 2 kilowords of RAM for buffers and other instrument data

Program code size was not estimated to be a major factor due to reuse of common routines between the instruments, so it was not specifically considered. The instrument sounds were to be synthesized at 44.1 kHz sampling rate.

It was clear that the available resources would place significant constraints on the synthesis algorithms and instrument design. The 16 bit multiplier meant that special consideration would have to be given to word length effects.

Chapter 2

Synthesis methods

A multitude of methods for digital sound synthesis have been conceived over the years. For extensive evaluation of them, see [50]. A number of common methods are presented in this chapter with emphasis on the more computationally efficient methods and especially subtractive synthesis and FM synthesis.

An ideal synthesis method would

1. be capable of producing realistic sounds
2. be easy to create new instrument sounds with
3. require little computation
4. require a small amount of operating memory (RAM)
5. require a small amount of constant storage memory (ROM)

In practice many of these requirements are mutually exclusive, so a compromise must be made. The qualities that are emphasized depend on the target application (realism, ease of new sound creation) and architecture (computation vs memory usage).

2.1 Additive synthesis

Any signal can be formed by summing a set of sine waves (partials) with varying amplitudes, frequencies and phase as shown in eq. 2.1 [9].

$$y = \sum a_n \sin(2\pi f_n t + \phi_n) \tag{2.1}$$

where a_n is the amplitude, f_n is the frequency and ϕ_n is the phase of each partial.

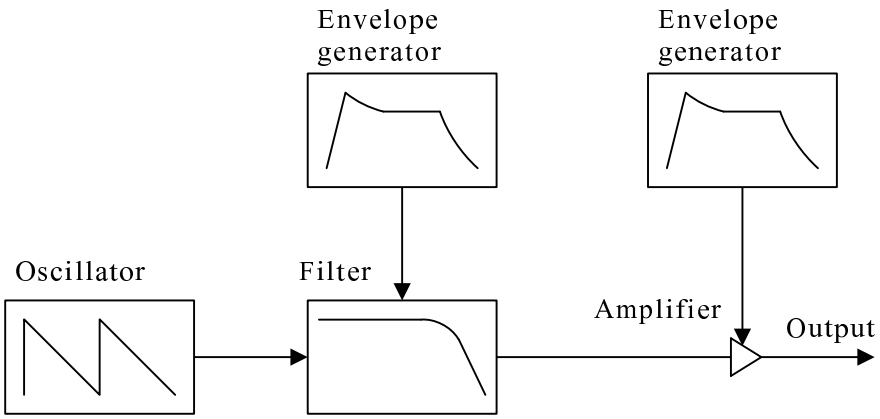

Figure 2.1: Block diagram for a typical subtractive synthesis instrument

While seemingly simple, the technique is complicated by the fact that real-life signals are time varying and not perfectly harmonic. Therefore the amplitudes, frequencies and phases are not constant. Storing them requires considerable storage, especially for attack portions where the variations are fast. A trivial implementation of additive synthesis also has high computational requirements. Some techniques have been developed to offset this which are based on using FFT [38].

Because of these complications, additive synthesis remains for the most part an academic technique and has not been widely used in practice.

2.2 Subtractive synthesis

Subtractive synthesis is a special case of Source-Filter synthesis. Source-Filter synthesis uses a sound source that is then processed by a filter. A typical example is modelling of human voice.

Subtractive synthesis replaces the source with one or more oscillators. Each oscillator can have different pitch, amplitude and waveform. The waveforms are typically simple geometric shapes, such as sawtooth or square. The filter is also often a simple lowpass shape with resonance although bandpass and highpass filters are also used. The filter is followed by an amplifier for adjusting the output amplitude. In addition to oscillators and filters and amplifiers, additional modulation blocks are used. These do not directly modify the signal but instead control some parameter of signal processing blocks. All the processing blocks are reviewed in more detail in the next subsections. Figure 2.1 shows a block diagram of typical subtractive synthesis instrument.

Subtractive synthesis was the operating method of analog synthesizers and still enjoys

wide popularity in the form of Virtual Analog Synthesis [46]. The simplicity of subtractive synthesis means that producing very realistic sounds is hard, but it can produce caricatures that, while lacking in realism, do sound quite pleasant. It also means that CPU usage and memory requirements are low. Additionally, the small number of parameters means that each parameter's effect can be heard and sound design is relatively easy.

2.2.1 Sound processing blocks

Oscillators usually produce simple waveforms that are rich in harmonics (for example sawtooth and square) but they can also use more complex waveforms or samples. The pitch of oscillators depends on played note but can optionally be also controlled by an envelope or LFO. Methods for making oscillators are discussed in section 3.

Filters are usually simple lowpass filters with varying resonance peak but other response shapes are also used. They are usually time varying, meaning that cutoff changes with time (usually controlled by an envelope and/or note pitch). Sometimes more than one filter is used in the signal path. Filters are discussed in more detail in section 4.

Amplifiers are simply gain blocks, with the gain controlled by an envelope. They can be implemented with a simple multiply operation.

2.2.2 Modulation blocks

Envelope generators produce a control signal to modify some aspects of the sound as the note progresses such as the amplitude or brightness. The envelope shape can vary, but a common one is the simple Attack-Decay-Sustain-Release (ADSR) shape shown later in section 5.5.1 figure 5.4. Typically both the amplifier and the filter have their own envelope and there may additionally be other envelopes to control for example oscillator pitches.

Low frequency oscillator means any oscillator that runs at very low frequency (below audio rate) and is used to control some synthesis parameter. They are typically used for producing vibrato or slow change in timbre of the sound. Typical LFO waveforms are sine and triangle wave. An LFO used for pitch vibrato often has an associated ramp envelope to smoothly introduce the vibrato to better mimic realistic playing styles.

Additionally, some static modulation sources are used. Note pitch often influences filter cutoff in addition to oscillator pitch. It can also affect envelope times and / or LFO frequency. Note velocity commonly controls the final output amplitude but can also affect the filter envelope amount.

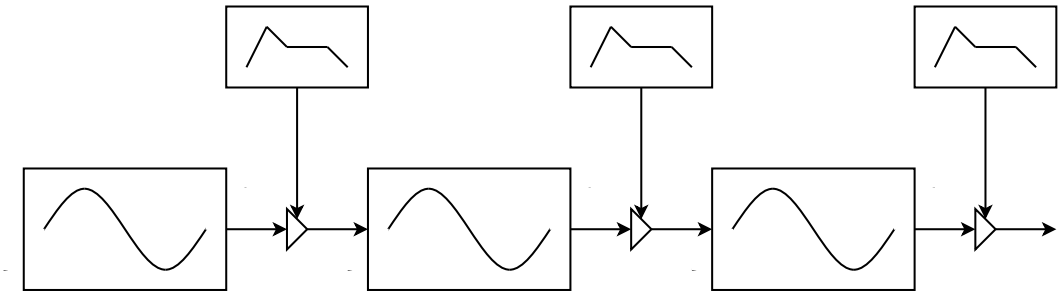

Figure 2.2: Block diagram of FM synthesis

2.3 FM synthesis

FM (Frequency Modulation) synthesis [10] uses a set of oscillators, often called operators, that act as modulators and carriers. FM synthesis is an extension of the similarly called radio transmission method to audio synthesis. Unlike in radio transmission, where the carrier is a very high frequency sinewave and the modulator is the sound signal, in FM synthesis the carrier and the modulator frequencies are of similar magnitude.

A carrier is an operator that produces the final audio output. A modulator is any operator that is connected to the frequency input of another operator. In addition to the oscillator, an operator typically has dedicated amplitude and / or frequency envelope. The operators can be arranged in different configurations, called *algorithms*. *Feedback FM* [51] is an extension to normal FM where the output of an operator is fed back to its input. Figure 2.2 shows a block diagram of a typical FM patch.

The equation for a simple one modulator one carrier FM system is

$$y(n) = A(n)\sin(2\pi f_c n + Isin(2\pi f_m n)) \tag{2.2}$$

where $A(n)$ is the amplitude, f_c is the carrier frequency, I is the modulation index and f_m is the modulator frequency. With feedback FM this becomes

$$y(n) = A(n)\sin\left(2\pi f_c n + I\sin(2\pi f_m n) + I_{fb}\, y(n-1)\right) \tag{2.3}$$

where I_{fb} is the feedback amount.

The spectrum is determined by the ratio between carrier and modulator frequencies and the modulation and feedback amounts. The spectrum can be calculated by the use of Bessel functions. This is rather complex and gets worse when more operators and time varying control is used. The reverse is unfortunately not possible, so FM sounds must be created by trial and error, although there has been some recent work in this [18, 22]. Figures 2.3 and 2.4 show example waveforms and spectra for the simple three operator FM configuration as shown in figure 2.2. Modulation index was 0.333 for both second and third operators.

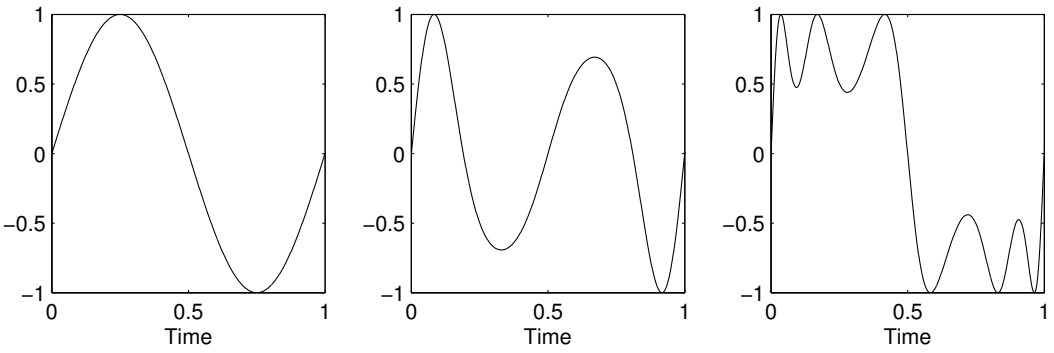

Figure 2.3: Example of three operator FM waveforms

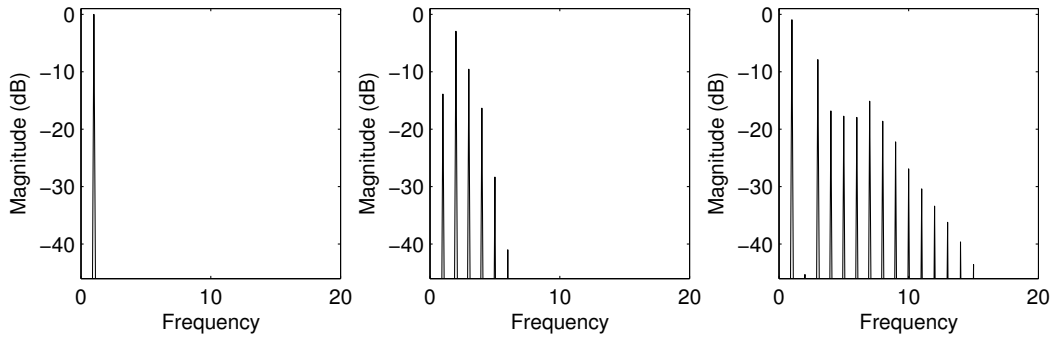

Figure 2.4: Example of three operator FM spectra

The ratio between carrier and modulator frequency determines the spacing of the partials and must be integer for the sound to be harmonic. The modulation index and feedback amount determine the "brightness" of the sound.

In practical systems four to six oscillators are used, each having a dedicated frequency and amplitude envelope. As each oscillator can have up to two envelopes, this results in a very large number of parameters. Since the spectrum can't be easily predicted, replicating existing sounds with FM requires a lot of work.

The low computational requirements of FM have caused it to be used in a number of commercial products with Yamaha DX-7 [58] bringing it to popular usage. Many people feel that the sound produced by FM synthesis is rather "cold" or "sterile". This, coupled with the difficulty of designing instruments has led to decline in use of FM synthesis in the recent years. FM still excels at producing inharmonic bell-like sounds though.

2.4 Sampling

In sampling, recordings of relatively short sounds are played back [37]. The playback speed is varied, depending on the required pitch, and often some kind of amplitude envelope is also applied. Because instrument timbre is often different for different pitches, several samples are used with the choice of sample depending on the played note and / or velocity.

Because the sample length is several seconds, the memory requirements for sampling are very large. Using some form of compression, such as ADPCM [26], can lower the memory requirements but this can usually only result in 2-4 times reduction in memory. Another option is to loop the sustain part of the sample. As the sustain part of sounds is usually relatively static, a small portion of it can be repeated over and over without affecting the perceived sound quality too much. While usage of compression and looping can reduce the memory requirements quite much, sampling still requires much more memory than most other synthesis methods.

To allow different pitches to be produced, the sample playback rate must be variable meaning the sample must be resampled on the fly. The simplest way is to advance at a slower / higher rate in the sound sample and pick the nearest position at each sample point. This results in poor quality though so some form of interpolation is almost always used.

To achieve reasonable quality, some interpolation method must be used. In *Sinc* interpolation, a polyphase filter is evaluated for each sample point [49, 14]. This results in very high quality at the cost of relatively heavy computational load. Another option is to use some polynomial interpolator [33]. The computational cost is less but the quality is also lower. In some cases simple linear interpolation between adjacent sample indices is enough. The choice of interpolation method depends on the required quality and the implementation

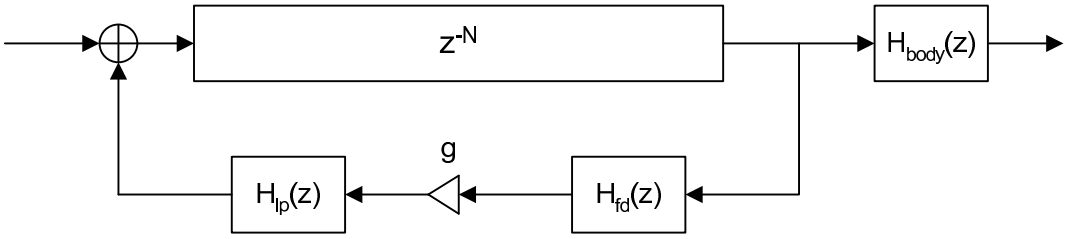

Figure 2.5: Simplified diagram of physical string model

architecture.

Sampling can produce very good quality synthesis if the sound set is large enough. This, coupled with the decreasing cost of memory, has led to sampling being the most common synthesis method used in synthesizers nowadays.

2.4.1 Wavetable synthesis

Wavetable synthesis can be thought of as a special case of sampling. A short section (often a single period) of sample is played in a loop [1, 6]. When several sections are stored in a table and the table index is swept, the timbre can be dynamically changed. This was pioneered in the PPG Wave synthesizer. Another common use is storing multiple versions of the same waveform that differ for example by the number of harmonics present.

2.5 Physical modelling

Physical modelling (see for example [44] or [57]) is a group of methods which have in common the fact that they produce sound by imitating the behaviour of real acoustic instruments. As such, they can also model the interaction with player better than other models, allowing them possibly to produce the most realistic sound of any synthesis method. Unfortunately this interaction cannot be easily taken advantage of in conventional synthesis frameworks where input is from a MIDI file due to the limited input possibilities. Nonetheless, given a high quality model of an instrument, physical modelling can produce very high quality and realistic sounds.

Waveguide physical modelling methods are based on a delay loop with feedback and frequency dependent damping. A simplified diagram of a string model is shown in figure 2.5. The model has a fractional length delay line z^{-N}, that determines the pitch of the sound. In the feedback loop, there is a fractional delay filter $H_{fd}(z)$, followed by a lowpass filter, that implements frequency dependent damping and then feedback gain that determines the

decay time of the note. Optionally there may be an all-pass filter that adds inharmonicity to higher partials (not shown). The input (or initial contents) to the delay line is the excitation pulse. The output of the model is fed through a filter $H_{body}(z)$ that simulates the body response of the instrument, but this can often be combined with the initial excitation signal.

Physical modeling usually only needs memory for the delay lines and body impulse response, which is considerably less than that required by sampling. The computational complexity is higher than sampling but not overly much for the simpler models. The main challenge is determining the model parameters and the body response [54]. Physical modeling was pioneered in commercial use by the Yamaha VL-1 synthesizer [42].

2.6 Other methods

A number of other synthesis methods have been invented over the years, but they have not gained widespread popularity. Some of the more notable ones for instrument synthesis are waveshaping, phase distortion and granular synthesis.

2.6.1 Waveshaping

Waveshaping [7] [2] takes as input a simple waveform and applies a nonlinear shaping function to it

$$y = f(w(t)) \tag{2.4}$$

where $w(t)$ is the input waveform, f is the shaping function and y is the resulting output. If a sine wave is used as the input and a Chebyshev polynomial as the shaping function, the resulting spectrum can be controlled exactly. Figure 2.6 shows several waveshaping functions and the result for a sine wave input for each.

2.6.2 Phase distortion

Phase distortion is somewhat similar to FM synthesis. Instead of a modulator input, the phase of an oscillator is varied with a preset curve [25]. Figure 2.7 illustrates the method using piecewise linear curve and sine oscillator. This method was used in some Casio synthesizers [19].

2.6.3 Granular synthesis

Granular synthesis divides a sound into small fragments, called *granules* [37]. The granules are then repeated at a different rate and overlapped, allowing independent scaling of pitch and timbre.

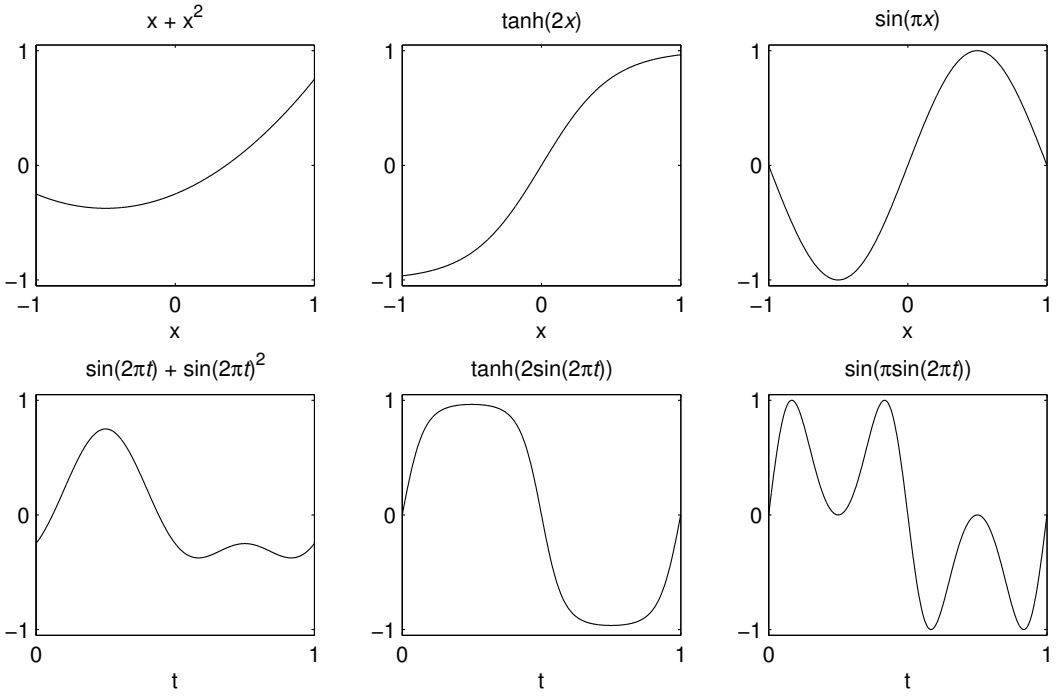

Figure 2.6: Example waveshaping functions

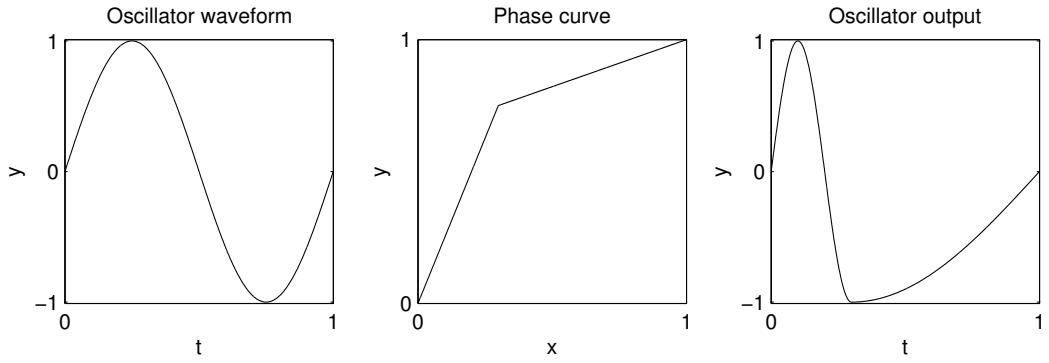

Figure 2.7: Example of phase distortion

Chapter 3

Oscillator algorithms

Subtractive synthesis relies heavily on spectrally rich waveforms. These are usually simple geometric shapes such as the sawtooth, pulse, triangle and sine waves shown in figure 3.1. These are presented in section 3.1.

The main challenge is to find computationally efficient algorithms for generating such periodic waveforms that don't suffer from excessive aliasing. Generating geometric waveforms trivially is equivalent to sampling the waveforms without any form of bandlimiting. Unless the spectrum falls very fast, this results in heavy aliasing, especially at high frequencies. Figure 3.2 compares the spectrum of aliasing and bandlimited sawtooth waves. Algorithms for reducing or eliminating this aliasing are presented in section 3.2. The em-

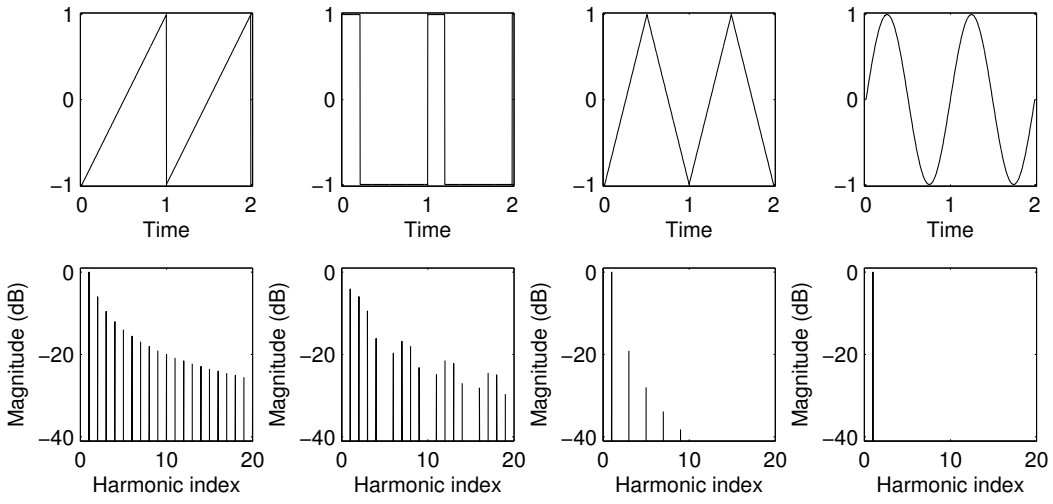

Figure 3.1: Oscillator waveforms (Sawtooth, Pulse, Triangle, Sine) and their respective spectra

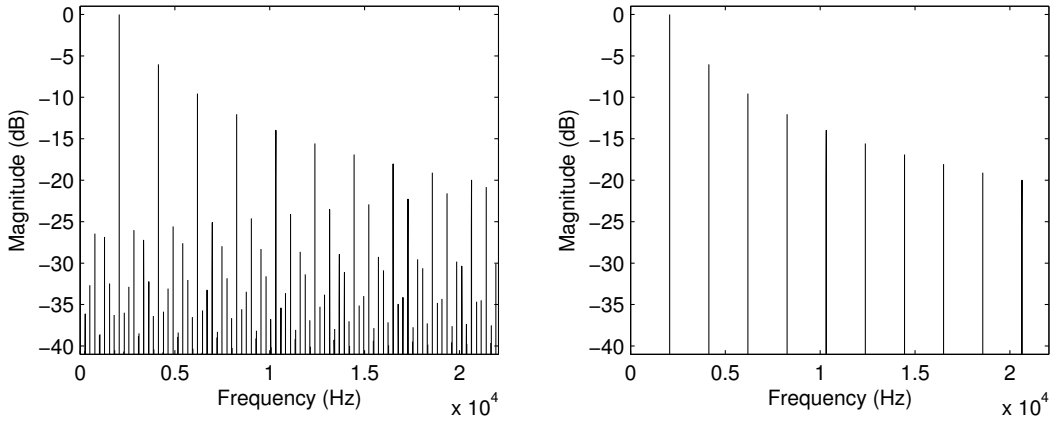

Figure 3.2: Spectrum of trivial sawtooth (aliasing) and bandlimited sawtooth (no aliasing)

phasis is on sawtooth generation, as the same techniques can be used to bandlimit most geometric waveforms. For a more in-depth review, see [56].

3.1 Waveforms

Traditionally subtractive synthesis has used simple geometric shapes both because they are simple to produce with analog circuits and because they are rich in harmonics and thus suitable source material for a filter to process. The most common waveforms are sawtooth, pulse, triangle and sine.

3.1.1 Sawtooth wave

Sawtooth waves are the most useful of these waveforms as they contain both even and odd harmonics. Additionally both pulse / square waves and triangle waves can be derived from sawtooth waves. Therefore most of this chapter concentrates on algorithms for producing sawtooth waveforms. The spectrum of sawtooth waveform falls at a 6 dB / octave rate.

A sawtooth wave can be produced trivially using a simple modulo-counter

$$Saw(n) = 2\left(\left(Saw\,(n-1) + \frac{f}{F_s}\right) \mod 1\right) - 1 \tag{3.1}$$

where n is the sample index, f is frequency in Hz and F_s sample rate.

3.1.2 Pulse wave

Pulse waves can be produced trivially by comparing the output x of a modulo-counter with the pulse width pw

$$Pulse(x) = \begin{cases} -1 & x < pw \\ 1 & x \geq pw \end{cases} \tag{3.2}$$

The spectrum of pulse waveform falls at approximately 6 dB / octave rate asymptotically and it thus aliases similarly to the sawtooth wave. Fortunately, the pulse wave can be produced by subtracting two delayed sawtooth waves.

$$Pulse(t) = Saw(t) - Saw(t - pw/f) \qquad 0 < pw < 1 \tag{3.3}$$

where f is the frequency in Hz, pw is pulse width and t is time in seconds. Therefore only the sawtooth waveform has to be antialiased. Another possible advantage with this method over naive pulse wave generation is that the DC-offset of the resulting waveform is zero [55]. Square wave can be considered simply a special case of pulse wave with pulse width set to 50%.

3.1.3 Triangle wave

Triangle wave can be trivially generated by taking the absolute value of the modulo-counter and then scaling and offsetting it.

$$Tri(x) = 2|x| - 1 \qquad -1 \leq x < 1 \tag{3.4}$$

where x is the output of the modulo counter. The spectrum of the triangle wave falls approximately 12dB / octave and so triangle wave can often be used as-is without any anti-aliasing. If anti-aliasing is required, triangle wave can be produced by integrating and scaling a bandlimited square wave

$$Tri(t) = f \int Sqr(t)dt \tag{3.5}$$

where f is the frequency in Hz and t is time. Another way is to directly bandlimit the discontinuity in the derivative. Some algorithms for producing bandlimited sawtooth wave can be altered to produce bandlimited triangle wave natively (section 3.2.5).

3.1.4 Sine wave

Sine wave generation methods can be divided into recursive and non-recursive. Non-recursive methods generally have a modulo-counter that is used either as index to a lookup table [32] or in a polynomial approximation [34]. Non-recursive methods are stable and

do not produce discontinuities when changing the frequency. On the other hand they have lower accuracy than recursive methods. Recursive methods [13] [23] use the previous output or intermediate values to calculate the next value. They are often very efficient but require relatively high numerical precision to avoid instability. Additionally many recursive methods produce discontinuities or require recalculating the state whenever the frequency changes.

3.2 Sawtooth algorithms

As stated in section 3.1.1, trivial sawtooths alias severely. Several algorithms have been developed to reduce or eliminate the aliasing. None of the algorithms are perfect for all situations and the best choice depends on the used platform and task. Generally desired features are

1. High reduction of aliasing
2. Low CPU usage
3. CPU usage independent of oscillator frequency
4. Low memory usage
5. Possibility to update oscillator frequency fast

Some of the algorithms allow adjusting the amount of alias reduction in exchange for less CPU or memory usage. For a recent review of the algorithms, see [56].

3.2.1 Additive sawtooth

Additive sawtooth simply means summing the individual harmonics of sawtooth waveform.

$$\sum_{n=1}^{N} \sin(2\pi f t)\frac{1}{n} \tag{3.6}$$

where f is frequency, t is time and N is the amount of harmonics

$$N = floor\left(\frac{F_s}{f}\right) \tag{3.7}$$

The amount of harmonics is large, growing as the frequency decreases. Therefore this method is not feasible except for very high frequencies and even then a fast method for generating sinewaves is required. Additive sawtooth generation is however useful as a comparison tool to provide a reference implementation.

3.2.2 Wavetable

One possibility for generating bandlimited sawtooth is to store the waveform in a table. Here a set of tables is used, with the current table determined by frequency so that for each wavetable

$$N \leq \frac{F_s}{f} \tag{3.8}$$

where f is frequency in Hz, N is the number of harmonics and F_s the samplerate. The number of wavetables depends on how large frequency range is covered and how often the wavetable is switched. One wavetable per octave, per note or per harmonic is typical.

According to Nyquist theorem, the table length M must be

$$M \geq 2N \tag{3.9}$$

This requires the use of a perfect interpolator, which is costly. The requirement can be alleviated by using a larger table than strictly necessary. This however increases memory requirements. Another option is to settle for lesser aliasing reduction. Niemitalo [33] has studied the performance of polynomial interpolators with varying amounts of oversampling.

Wavetable sawtooth generation satisfies requirement 3 and for certain table switching schemes and platforms also 5. Points 1, 2 and 4 are interdependent. Satisfying any one of them better causes one or both of the remaining two to be less well satisfied. The choice then depends on requirements and platform. For example on PC platforms memory usage is usually not an issue and so generous oversampling on the tables can be used to increase quality and lower CPU usage.

3.2.3 Differentiated parabole wave

Välimäki [53] has proposed a novel method to reduce the aliasing of a trivial sawtooth. The method takes advantage of the fact that it is possible to generate the sawtooth waveform by first generating a parabole waveform and then differentiating it. The parabole waveform spectrum falls at 12 dB / octave rate and thus aliasing is much reduced compared to the sawtooth wave even if no attempt is made at bandlimiting the parabole wave. Figure 3.3 compares the resulting aliasing from trivial sawtooth and sawtooth made by differentiating parabole wave.

Because the parabole wave does not need to be separately bandlimited, it can be generated by multiplying a trivial sawtooth wave with itself.

Differentiating and scaling the parabole wave results in sawtooth wave with reduced aliasing

$$Saw_{ra}(t) = c\frac{d}{dt}Saw(t)^2 \tag{3.10}$$

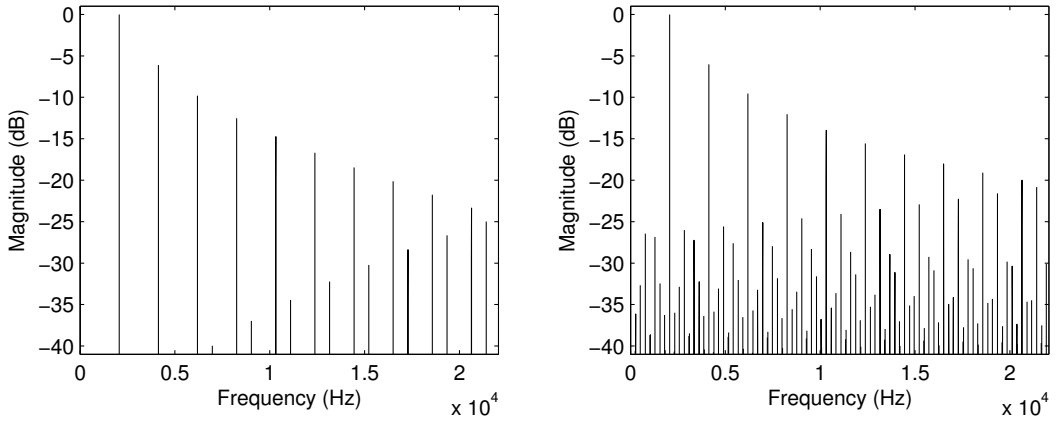

Figure 3.3: Spectrum of DPW sawtooth and trivial sawtooth

where c is a scale factor to compensate for the attenuation caused by differentiation

$$c = \frac{F_s}{8f(1 - \frac{f}{F_s})} \tag{3.11}$$

While DPW provides only modest reduction in aliasing, this is often sufficient in practice, as the aliases are most attenuated at low frequencies, where they would not be masked by the fundamental or overtones and would thus be most audible (as is the case with trivial sawtooth waveform). Using modest 2x oversampling will further increase the quality attenuating the first aliases further 18 dB.

The CPU and memory requirements are both very low, coming second only to trivial sawtooth. Apart from the modest reduction in aliasing, the possibly largest problem with the method is that the scale factor c is inversely dependent on the frequency, requiring a division whenever frequency is changed. This makes the method not well suited for situations where there is a need to rapidly change the frequency (in FM synthesis for example). However, if the frequency is only changed occasionally, this method has probably the best CPU usage / sound quality ratio. For architectures with low bitdepth the use of differentiation in the method may present problems for low frequencies.

3.2.4 Bandlimited impulse train synthesis (BLIT)

BLIT synthesis as described by Stilson and Smith [47] generates alias free waveforms by integrating a bandlimited impulse train. Sawtooth can be generated simply by integrating a DC-offsetted impulse train

$$Saw(t) = \int (BLIT(t) - C)dt \tag{3.12}$$

where $BLIT(t)$ is the impulse train and C is the DC offset of the impulse train. The subtraction is required to keep the integration from ramping off to infinity. In practice a leaky integrator has to be used to avoid ramping due to roundoff errors. The bandlimited impulse train can be generated algebraically

$$\mathrm{Sinc}_M(x) = \frac{\sin(\pi x)}{M \sin(\pi x/M)} \tag{3.13}$$

This method requires two $sin()$ evaluations and a division per sample, both of which are time consuming.

Another way to generate BLIT is to use a sum of windowed Sincs (BLIT-SWS). Here each unit impulse is replaced by a windowed $Sinc$ function, centered on the unit impulse position. In practice a set of windowed $Sinc$ functions is stored in a table for each fractional position and some form of interpolation is used between them at runtime. A modulo-counter can still be used for tracking the waveform period. Then a new $Sinc$ needs to be inserted whenever the counter rolls over. Whenever that happens, the fractional position can be calculated by

$$Pos_{frac} = \frac{xF_s}{f} \tag{3.14}$$

where x is the output of the modulo counter ($0 \leq x < 1$).

This is the same as lowpassing the ideal impulse train and then sampling it. Although the method is known as the Sum of Windowed Sincs, it should be noted that any lowpass filter can be used as long as it fits the required specifications for passband width and stopband rejection [56]. Therefore Remez exchange algorithm [35] or other optimization techniques [3] can be used to calculate the FIR filter and result in a decrease of the required filter length.

A general problem with the BLIT-SWS is that CPU usage is proportional to frequency (since an impulse is inserted once for each period). Some lookahead is also required since the $Sinc$ functions are centered on the impulse position and so a new $Sinc$ needs to be mixed in several samples (depending on the window length) before the actual impulse center. Another CPU increasing factor is the division for finding $Sinc$ phase that is required for each period. A benefit of BLIT is that it can be used to natively generate both square and triangle waves. Also the length of the window can be controlled and thus CPU usage can be traded for better quality and vice versa.

3.2.5 Bandlimited step synthesis (BLEP)

Brandt extends BLIT by using a minimum-phase impulse and pre-integrating it, producing a minimum-phase bandlimited step (MinBLEP) [4]. Although claimed, using a minimum-phase impulse does not entirely eliminate the lookahead of BLIT, but merely reduces it

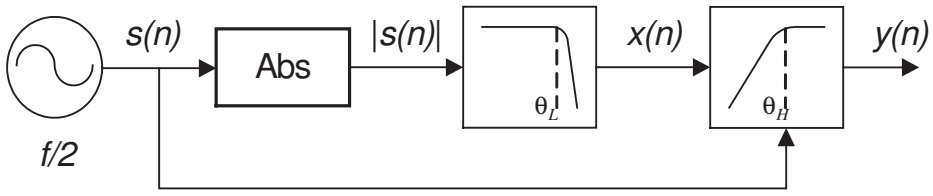

Figure 3.4: Block diagram for generating bandlimited sawtooth approximation by rectifying sine wave

[56]. While originally introduced for implementing hard sync, the method can be used to produce bandlimited version of any waveform with discontinuities as long as all derivatives of the waveform are continuous.

As a BLEP is preintegrated, the integration step of BLIT can be skipped and a BLEP pulse can simply be mixed with the non-bandlimited waveform any time there is a discontinuity. The fractional position can be calculated the same way as with BLIT (see eq 3.14).

The main advantage of BLEP over BLIT-SWS is the elimination of integration. This eliminates possible numerical accuracy problems, as leaky integrators, which are filters with very low cutoff frequency, are no longer needed. Performing the integration before sampling also tilts the spectrum by 6 dB / octave compared to integrating after sampling (as happens with BLIT). This reduces aliasing considerably. The CPU cost is still proportional to frequency and a division for each period is still required.

If the BLEP is further integrated, a bandlimited ramp (BLAMP) is produced [43]. This can be used to directly bandlimit triangle or any other waveform that has discontinuities in the first derivative. Using BLAMP in combination with BLEP allows better bandlimiting of hard sync, as each bandlimited derivative decreases the aliasing by additional 6 dB / octave. Numerical issues and diminishing returns limit the maximum derivative thus bandlimited to second or third derivative in practice.

3.2.6 Rectified sine

Lane et al. [23] present a method for generating a bandlimited approximation of sawtooth. While not generating exact sawtooth, the spectrum is very similar. The method uses absolute value of sine, followed by lowpass and highpass filtering (fig 3.4).

The lowpass filter is a “brick-wall” filter used to suppress high frequency aliasing components. The highpass filter is a tracking first order filter with the cutoff set higher than the sawtooth frequency so that the first N harmonics are boosted. An advantage of the method is that it can be used to natively produce square and triangle wave although these require

the use of another sinewave at twice the fundamental frequency [23].

While producing relatively good quality results, the use of filters makes the method CPU consuming. The tracking highpass filter means that the coefficients have to be recalculated whenever the sawtooth frequency changes.

Chapter 4

Filter algorithms

Filter design can be done by a variety of methods, a common one being starting from the analog prototype and then using the bilinear transform or other similar method to transform that to a digital filter. A good overview of digital filters and design methods is given in [45].

For time varying filters, other methods and filter structures are generally better suited. Filters used in musical audio processing can thus be divided into two categories: Generic filters, which usually have fixed frequency response and are designed using traditional methods, and musical filters, which are optimized for varying response and use special filter structures. The lowpass response is most common ín musical applications and is hence given most attention here.

4.1 Generic IIR filters

Generic IIR filters typically use Direct Form, Lattice or a similar traditional topology [45]. They can (in theory) have any desired frequency response shape. For computational reasons, higher order filters are usually split into multiple second and first order sections, connected either serially or in parallel. Generic IIR filters have relatively complicated and/or expensive design procedure and may have problems with stability when coefficients are changed fast. These problems limit their use in musical applications.

4.1.1 Second order sections

The coefficients for a second order section (often known as biquads) can be calculated directly for a given tuning and Q. The equations for common responses can be derived through the use of bilinear z-transform [17] or other means [24]. Parametric and shelving equalizers can also be calculated directly [5].

The transfer function of each second order section is of the form

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{4.1}$$

for a bilinear z-transformed lowpass filter the equations for coefficients are [17]

$$b_0 = 1/(1 + 2\zeta C + C^2) \tag{4.2}$$

$$b_1 = 2b_0 \tag{4.3}$$

$$b_2 = b_0 \tag{4.4}$$

$$a_1 = 2b_0(1 - C^2) \tag{4.5}$$

$$a_2 = b_0(1 - 2\zeta C + C^2) \tag{4.6}$$

where

f		cutoff frequency
ζ		damping factor
F_s		sampling frequency
C	$=$	$1/(\tan(\pi f/F_s))$

As can be seen, calculation of the coefficients is relatively complex and time consuming. Several multiplications, two divisions and one transcendental function evaluation are needed whenever cutoff frequency or resonance is changed. In musical applications, the filter often needs to be updated for every sample and recalculating the coefficients takes a large portion of the processing.

The sections can be implemented using any appropriate topology. A common one is the canonical second-order section (fig 4.1). It can be implemented by the difference equation

$$y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) - a_1 y(n-1) - a_2 y(n-2) \tag{4.7}$$

While this topology is very efficient, it does have problems with numerical precision at lower frequencies. Therefore other topologies, like lattice, may be better depending on the application and platform.

4.1.2 E-Mu Armadillo coding

Dave Rossum of E-mu Systems has presented a coefficient encoding method called Armadillo encoding [40] that allows efficient realtime interpolation of filter shapes.

A regular IIR filter is divided into second order sections

$$H(z) = c_0 \frac{1 + c_1 z^{-1} + c_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{4.8}$$

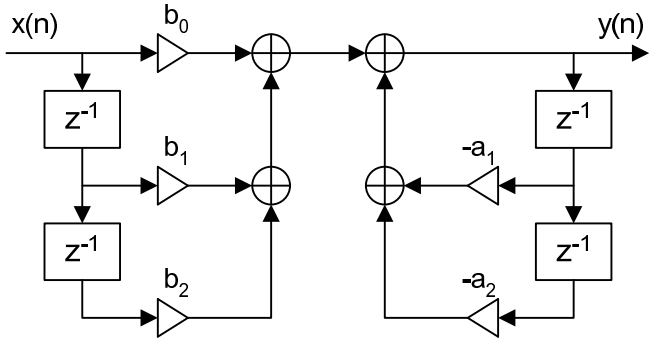

Figure 4.1: Direct form 1 second-order section

Numerator and denumerator can both be written as

$$p(z) = 1 + t_1 z^{-1} + t_2 z^{-2} \tag{4.9}$$

t_1 and t_2 can then be written as functions of v_1 and v_2

$$t_1 = -2 + 4 \cdot 2^{-v_1} + 2^{-v_2} \tag{4.10}$$

$$t_2 = 1 - 2^{-v_2} \tag{4.11}$$

v_1 can be shown to be approximately linear in a quantity called the musical octave number Ω

$$\Omega \approx -\frac{1}{2} v_1 + \log_2 \frac{F_s}{20\pi} \tag{4.12}$$

while v_2 is linear in peak / through height h in dB

$$h \approx 6 v_2 + 6 \tag{4.13}$$

Interpolating v_1 and v_2 allows smooth morphing between any two (or more) filter responses. The responses need not be traditional lowpass or highpass shapes but can instead be complex shapes such as parametric equalizers or shelving filters [16] for example.

A disadvantage of the coding is the requirement for fast exponential evaluation as four 2^x evaluations are needed per section per sample. If the response shapes are restricted and filter peak height (resonance) is held fixed, two of these evaluations can be removed. The method still likely remains unfeasible unless custom hardware is present for the exponent functions.

4.2 Musical IIR filters

Several structures have been devised that strive to both decouple the Q value from tuning of the center/cutoff frequency and provide fast update of the filter coefficients. They are

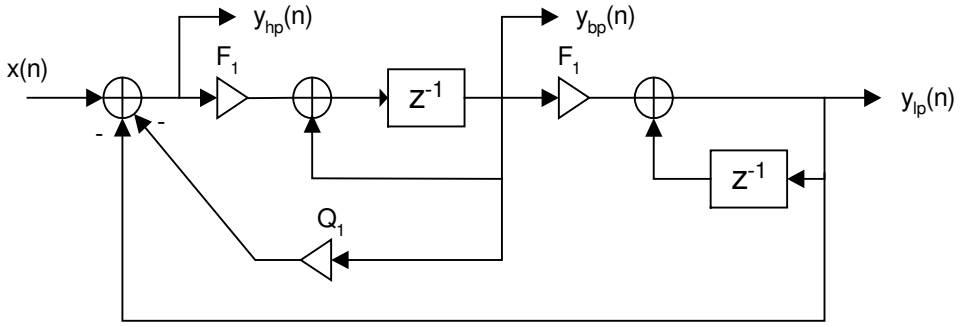

Figure 4.2: Chamberlin SVF topology

usually based on the analog filter structures used in analog synthesizers. This allows use of extremely high Q values and even controlled instability (self oscillation) while being musically useful [20]. Two commonly used structures will be presented here and an entirely new structure based on the Korg MS-20 lowpass filter will be introduced.

4.2.1 Digital state variable filter

Chamberlin [8] presents a design based on the analog State Variable Filter (SVF). It is a second order filter with independently controlled tuning and Q that provides simultaneous lowpass, bandpass and highpass outputs. The filter structure is shown in figure 4.2.

Equations for coefficients F_1 (tuning) and Q_1 (Q) are

$$F_1 = 2\sin\left(\pi\frac{f_c}{F_s}\right) \tag{4.14}$$

where f_c is the cutoff frequency and

$$Q_1 = 1/Q \qquad 0.5 \leq Q \leq \inf \tag{4.15}$$

The transfer function is

$$H(z) = \frac{r^2}{1 + (r^2 - q - 1)z^{-1} + qz^{-2}} \tag{4.16}$$

where

$$r = F_1 \qquad q = 1 - F_1 Q_1 \tag{4.17}$$

The simple tuning and Q equations make the SVF filter an attractive choice for subtractive synthesis. Although exact tuning requires the use of a tuning table, this table is sine shaped and likely to be already present in memory. The availability of simultaneous responses is also an advantage. Disadvantage is that the filter becomes unstable [12] when $F_1 \geq 2 - Q$,

limiting the usable tuning range. To reach full tuning range at all values of Q, oversampling must be used.

The state variable filter can also be used as a low frequency sine oscillator by setting Q infinite [13]. The roundoff errors cancel each other on most platforms, making the oscillation stable.

4.2.2 Digital Moog lowpass filter

The Moog ladder filter [31] and other similar cascaded RC-section filters are common in analog synthesizers. The filter is a fourth order (24 dB / oct) filter with controllable tuning and Q (resonance). Stilson and Smith have developed a digital implementation of this filter [48]. Like the analog filter, the digital implementation consists of four one-pole filters in series (fig 4.3). Global negative feedback is used to introduce a resonance peak near the filter cutoff frequency.

To produce a realizable digital filter, a unit delay has to be inserted in the feedback path. This makes tuning and Q coupled and defeats the main point of the original filter. Therefore it is of interest to compensate for the coupling. Stilson and Smith [48] show several versions for the compensation by using separation table and modifying the one-pole filter structures. Probably the most interesting is their "Compromise" version

$$G_1(z) = \frac{(p+1)}{1.3}\frac{z+0.3}{z+p} \tag{4.18}$$

$$G(z) = \left(\frac{(p+1)(z+0.3)}{1.3(z+p)}\right)^4 \tag{4.19}$$

where

$$p = e^{-2\pi f/F_s} \tag{4.20}$$

Moving the zeroes to position -0.3 on the z-plane real axis causes Q to be almost completely independent of cutoff frequency. Use of a tuning table or polynomial is still required if exact tuning is required at high Q.

A nonlinearity should be placed within the feedback loop to limit the amplitude of oscillation and to prevent numerical overflow if using very high Q. Huovilainen [20] has implemented a digital model of the nonlinearities present in the original analog circuit and a related version where the nonlinearities have been folded into a single nonlinearity inserted before the first filter stage [55].

One issue with the Moog filter that may be a problem in some applications is that the passband gain decreases when resonance is increased (as resonance is increased by increasing negative feedback). This can be corrected by subtracting a portion of the input signal from the feedback signal before scaling the feedback by resonance amount [11] [55]. The

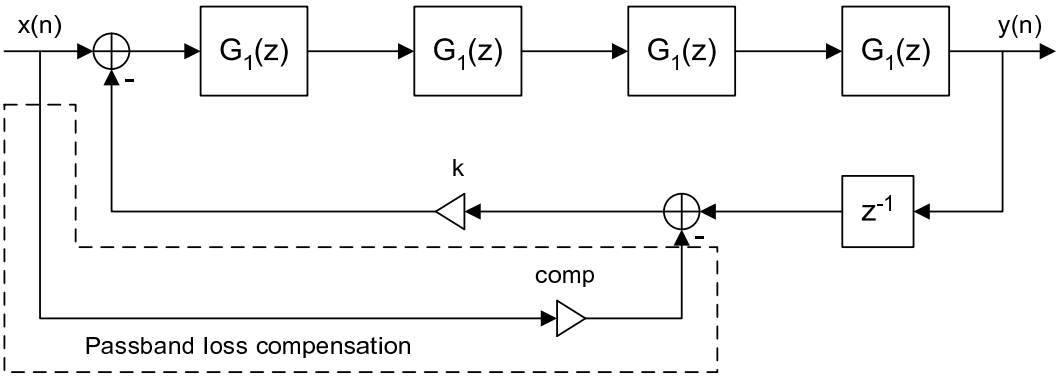

Figure 4.3: Digital Moog filter structure

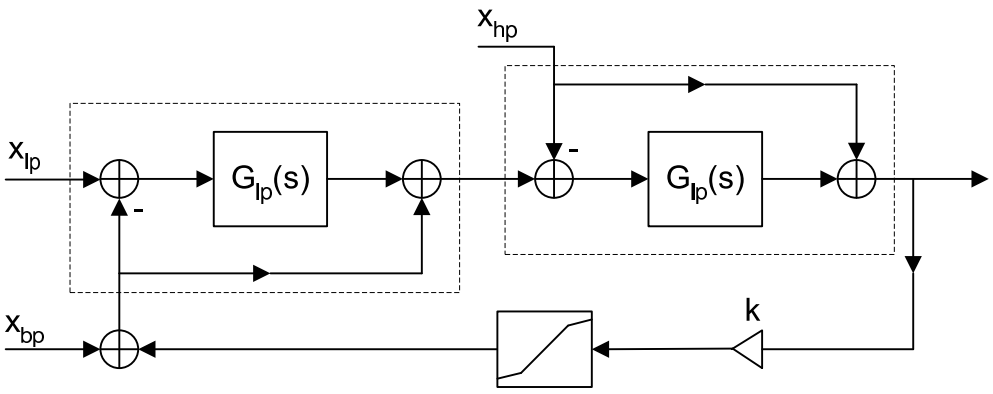

Figure 4.4: MS-20 filter structure

amount of passband loss can be adjusted between 12 dB and 0 dB by changing the portion of input that is subtracted. A 6 dB loss, achieved by subtracing half of the input signal ($comp = 0.5$), is a good default value.

4.2.3 MS-20 filter

Figure 4.4 shows the structure of the voltage controlled lowpass filter used in the Korg MS-20 analog synthesizer [21].

The filter consists of two first order voltage controlled lowpass sections, adjustable feedback k and a diode clipper for limiting the amplitude of the feedback. Second order lowpass and bandpass and first order highpass responses can be realized from the filter by varying the point where the signal to be filtered is inserted. Highpass response is realized by summing the input with inverted and lowpassed signal (the second dashed box). Similarly,

bandpass response is accomplished by configuring the first section as highpass filter and the second as regular lowpass filter.

Adjustable resonance is implemented by feeding back a portion of the output signal to the bandpass input. Because the bandpass response is realized by cascading first order highpass and lowpass filters, the phase shifts of the filters cancel to zero and the feedback is therefore positive at the cutoff frequency. As both sections have a gain of -3 dB at the cutoff frequency, setting the feedback gain k higher than 2.0 will result in the filter self oscillating. The diode clipper lowers the gain to about one fourth once the signal exceeds the threshold voltage of the diodes, thus limiting the amplitude of the oscillation.

Digital MS-20 filter

The MS-20 filter can be discretized by keeping the filter topology the same as in the analog version and replacing both first order sections by the digital equivalents. Using bilinearly transformed sections would keep the phase and amplitude response same at the cutoff frequency. However, keeping the resonance path as-is would result in a delay free feedback loop, making the filter unrealizable. Inserting a unit delay in the feedback path makes the filter realizable, but the extra phase shift compromises the zero phase shift positive feedback.

We thus opt to try replacing the first order sections by both bilinearly transformed sections and trivial digital one-pole sections. The resulting structure (including only the lowpass input) are shown in figures 4.5 and 4.6. Both versions require five multiplications and four additions per sample, as shown, making them very computationally efficient. Trivial transformations allow replacing two of the multiplications in the trivially discretized version with simple additions.

The non-linearity present in the resonance path is preserved in the discretized filter versions. The circuit uses a non-inverting opamp gain stage with clipping diodes inserted in parallel to the feedback resistor. Once the signal exceeds the threshold of the diodes, they softly limit the gain to one. Digital modeling of diode clipping has been studied in more detail in [59]. A reasonable first order approximation of the diode clipper is to use a piecewise linear function to lower lower the gain to one fourth when the signal exceeds a set threshold value. The $f_{clip}(x)$ function is then

$$f_{clip}(x) = 0.25x + 0.75sat(x, thres, -thres) \tag{4.21}$$

where $sat(x, thres, -thres$ is a saturator with adjustable threshold. The clipper is similar to the simplified non-linear Moog filter presented in [55], although not as smooth.

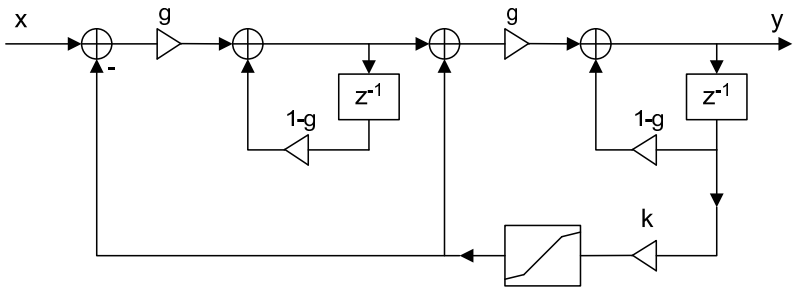

Figure 4.5: Trivial one-pole discretized MS-20 filter

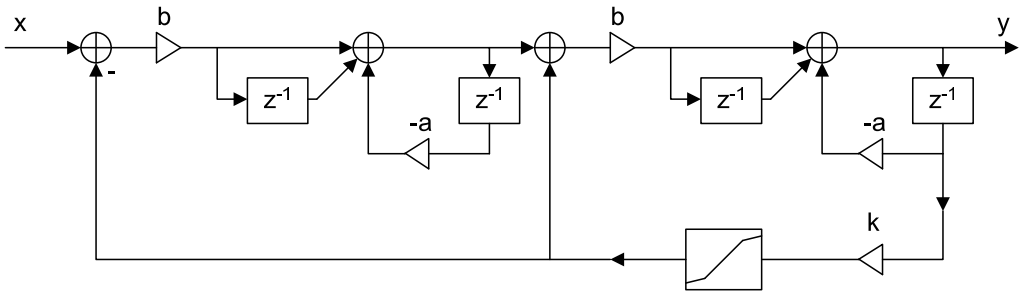

Figure 4.6: Bilinear transform discretized MS-20 filter

The filter coefficients can be calculated as

$$g = \exp\left(-2\pi \frac{f}{F_s}\right) \tag{4.22}$$

for the trivial one-pole discretized version and

$$K = \tan\left(\pi \frac{f}{F_s}\right) \tag{4.23}$$

$$b = \frac{K}{K+1} \tag{4.24}$$

$$a = \frac{K-1}{K+1} \tag{4.25}$$

for the bilinear transform version.

Figures 4.7, 4.8, 4.9 and 4.10 show the frequency responses of the discretized filters for feedback gains of 0, 1.0 and 1.5 while sweeping the cutoff at octave intervals. The effect of inserting the unit delay can be clearly seen in how the amplitude of the resonant peak is no longer independent of the cutoff frequency.

Comparing the four sets of figures, it's immediately apparent that the bilinear transform discretized filter has superior response. For the highpass case, the passband gain of trivial one-pole version drops with increasing cutoff, rendering it unusable without some form

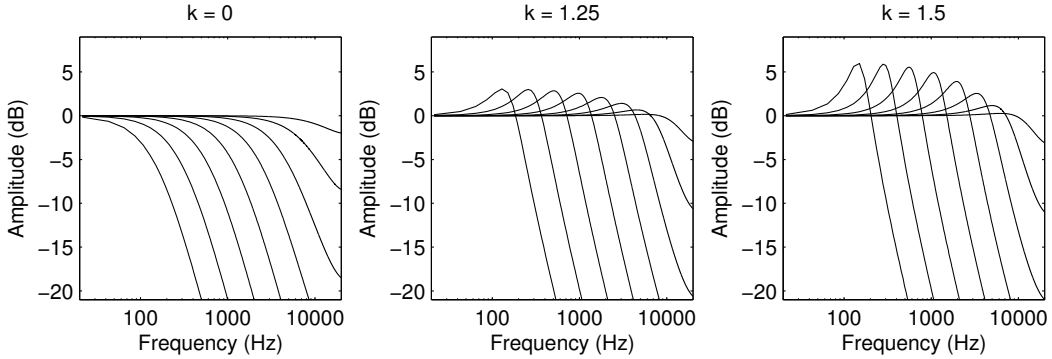

Figure 4.7: Trivial one-pole discretized MS-20 filter, lowpass response

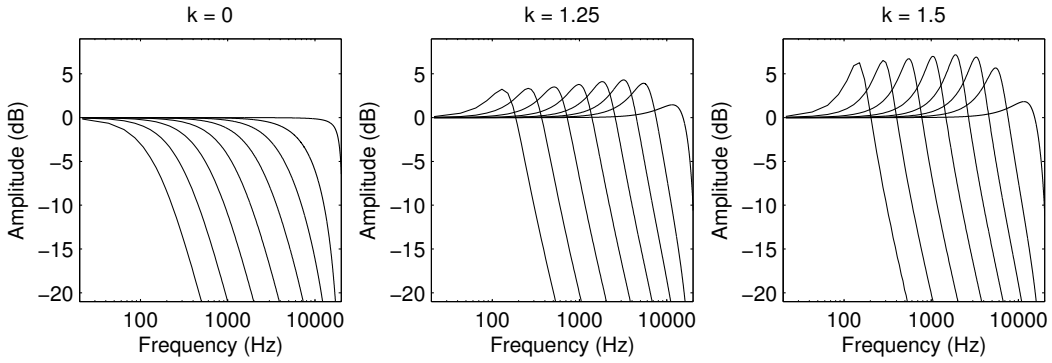

Figure 4.8: Bilinear transform discretized MS-20 filter, lowpass response

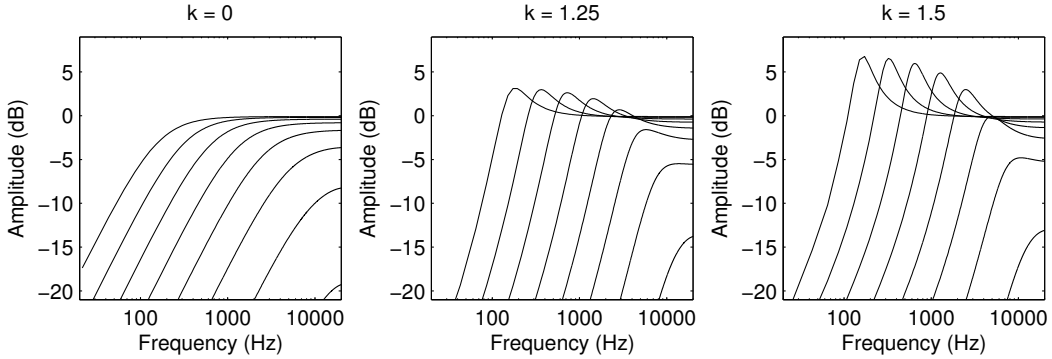

Figure 4.9: Trivial one-pole discretized MS-20 filter, highpass response

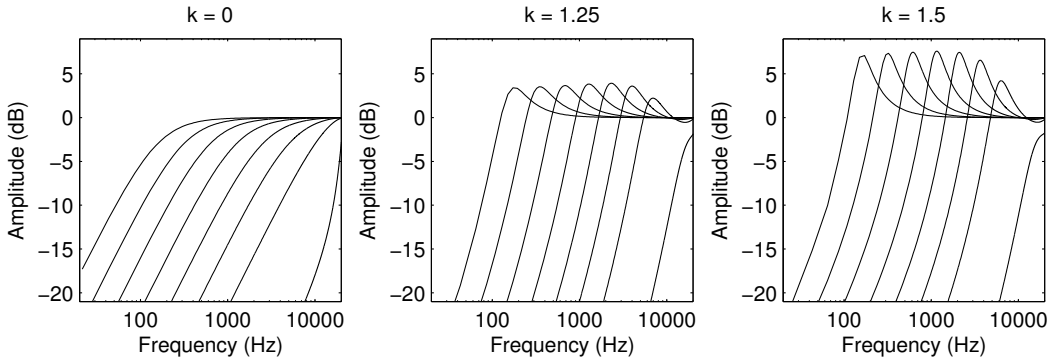

Figure 4.10: Bilinear transform discretized MS-20 filter, highpass response

of compensation. The increasing peaks for medium cutoffs in the bilinear transform discretized filter mean that it is no longer always stable for feedbacks of less than 2.0 as like the analog prototype. Luckily the increase is small and can be easily compensated with suitable cutoff dependent scaling of feedback gain. The trivially discretized filter version can still be useful for applications where the varying resonance can be compensated with other means or where the computational cost is critical. It requires fewer multiplications and only one interpolated coefficient, making it more efficient on some architectures.

Chapter 5

Implementation of synthesis algorithms

Subtractive synthesis (2.2) was selected as the main synthesis method to be used. It was deemed particularly suitable for implementation on the target platform and the relatively small number of parameters was also a benefit. Some melodic instruments are not well suited for it, though, and thus FM synthesis (2.3) was chosen for those.

A number of synthesis blocks were designed and implemented in MATLAB [27] and are described in this chapter. Implementing the actual DSP code was out of the project scope and will not be presented. Additionally some synthesis blocks were already present in the DSP platform and their exact details will be omitted.

Some consideration is given to low bit resolution issues in the algorithms, but detailed analysis was deemed to be out of the scope of the project. Such analysis would require knowledge of the exact details of the implementation platform and of the final program code.

5.1 Frame-based implementation

To increase the efficiency, a frame-based implementation was chosen. The sound is be generated in frames of N samples, so that each synthesis block produces that amount of samples as output for one procedure call. This allows more efficient code, as constants only have to be initialized once per frame, and DSP register usage is more efficient.

Another advantage is that modulation calculations would be performed at a reduced rate (once per frame) decreasing their cost to negligible. Within each audio processing block, the parameters would be calculated only for frame beginning and end and linearly interpolated in between. Some parameters, such as oscillator frequency or pulse oscillator phase offset,

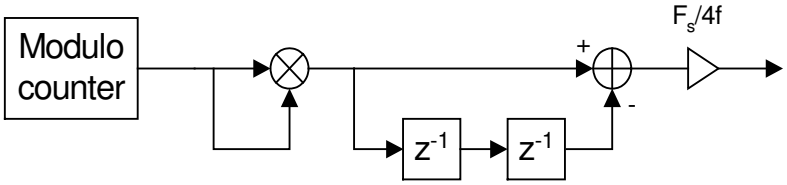

Figure 5.1: Diagram of DPW the sawtooth wave implementation

would not necessarily have to be interpolated at all.

The choice of frame length N was left to the producers of program code. As the time resolution of notes would be limited to N/F_s, N should be selected so that the time quantization is not audible. Less than two milliseconds is suggested.

5.2 Oscillators

The limitations of the target DSP platform place severe constraints on allowed memory usage and computational complexity for the waveform generators. The selected method should be as cheap as possible while still producing acceptable sound quality. The DPW (3.2.3) method requires only two multiplies and a single addition per sample while producing mostly audible alias free sawtooth. It was therefore selected for producing sawtooth and pulse waveforms with the implementation issues explained in the following two subsections. The DSP already contained a sine table and that removed the necessary to design sinewave algorithm. Triangle was produced with the trivial method, as the aliasing is already low enough. Some sounds also required the use of custom waveforms which is discussed in 5.2.3.

Implementation on a 16-bit DSP presents some problems for the algorithms. If a 16-bit accumulator is used for the modulo-1 counter, then the frequency resolution is only 0.67 Hz, which produces an error of 40 cents for the lowest note of piano (27.5 Hz). Therefore a 32-bit accumulator should be used for all oscillators.

5.2.1 Sawtooth

Implementation for the DPW sawtooth algorithm is shown in figure 5.1. The output of a modulo-1 counter is squared (multiplied by itself) and the square signal is mixed with a version delayed for two samples. Finally the output is scaled to compensate for the frequency dependent amplitude due to differentiation. The algorithm requires two additions and two multiplies per sample (assuming the frequency stays constant).

Increasing the differentiator delay from one to two samples is equivalent to convolving

the output of the DPW oscillator with a two-sample box filter. This reduces the amplitude of the partials in the highest octave and was found necessary to remove audible warbling that was otherwise present [55]. We speculate that the source of the warbling is inadequate attenuation of aliases near the nyquist frequency (see 3.3. Depending on the note, the aliases can fall quite close to the wanted harmonics, which can cause audible beating. Two sinewaves close in frequency and with equal amplitude are equivalent to one sinewave that is amplitude modulated at half the difference frequency.

The differentiation inherent in the algorithm presents a problem on low word length architectures. The peak amplitude of the output of the differentiator depends on the fundamental frequency so that $A = 4f_0/F_s$ (the exact peak amplitude is $4f_0(1 - f_0/F_s)/F_s$, but the approximation error is minimal [55]). For the lowest note of piano this gives a maximum output of +- 82 on a 16-bit architecture, which is roughly equivalent to 7-bit resolution. The situation would be even worse if the extra delay in the differentiator had been omitted, which would have halved the resolution. Another effect of the extra delay is a decrease in high frequencies (the effect is the same as filtering the signal with a two sample long box filter). Because the signal is always lowpass filtered at a further stage, this effect was deemed not to be a problem.

A second problem is the output scaling itself. Scaling by a fractional number larger than 1.0 is problematic on many DSPs, especially if programmed in some high level language like C. The problem was solved by first shifting the input left depending on the octave and then scaling by a fractional number below 1.0. This results in somewhat worse signal to noise ratio but is not critical as the signal will be lowpass filtered afterwards.

When using more than one oscillator in a sound, the oscillators are usually scaled before being mixed together. This scaling can be integrated with the frequency dependent output scaling, thus saving one or more multiplies per sample.

5.2.2 Pulse

As said in section 3.1.2, pulse waveform can be produced by subtracting two sawtooth waves with time-offset. Figure 5.2 shows the implementation. As shown, the pulse oscillator uses two sawtooth oscillators but with the difference that the second sawtooth does not have its own modulo-counter. Instead an offset is added to the first counter's output and advantage is taken of the wrapping in the two's complement addition operation. The offset is $offset = 2(pw - 0.5)$, where pw is the pulse width (0..1).

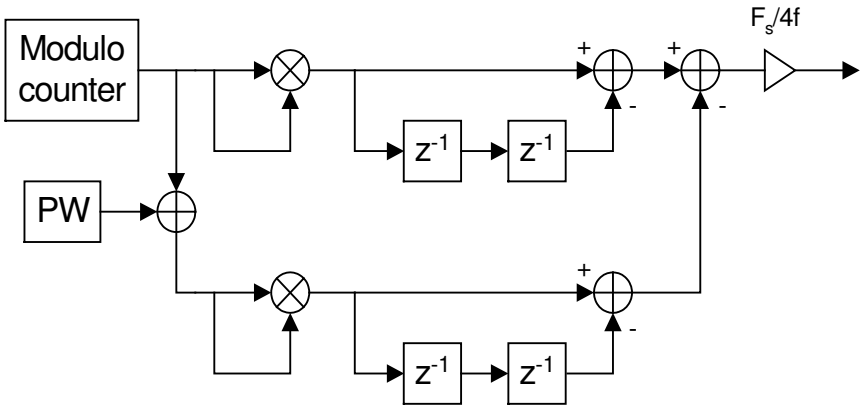

Figure 5.2: Diagram of the DPW pulse wave implementation

5.2.3 Wavetable

Some instruments have a spectrum that is relatively complex but changes in a simple predictable way as the sound progresses. Wavetable oscillator using arbitrary waveform can be used to recreate some of these. The spectrum change can be produced by having two detuned copies of the same waveform (to simulate beating) and by filtering the signal with a normal time-varying filter (to simulate decay of high harmonics).

A single cycle of the waveform is stored in memory. If memory constraints allow, several copies of the waveform can be stored and the used waveform selected at runtime depending on the played note. The cycle should be resampled to be exactly 2^M samples long so a modulo-counter (similar to the one used in sawtooth and pulse waveform generation) can be directly used to address the waveform. Linear interpolation should be used to avoid distortion. Another simple way to increase SNR is using longer length waveform than Nyquist criteria would require [33].

5.3 Filters

Filter algorithms should also have low computational complexity. This includes simple update of parameters such as cutoff frequency and resonance. Additionally the algorithms should be stable when using low precision coefficients and accumulators. Exact frequency response is not a requirement, as end users cannot access any parameters, and each instrument can be individually tailored to work within the required range. The digital MS-20 filter was selected for its very low computational demands, simple updating of parameters and stability. Some instruments also use static low order IIR filters. These use pre- existing

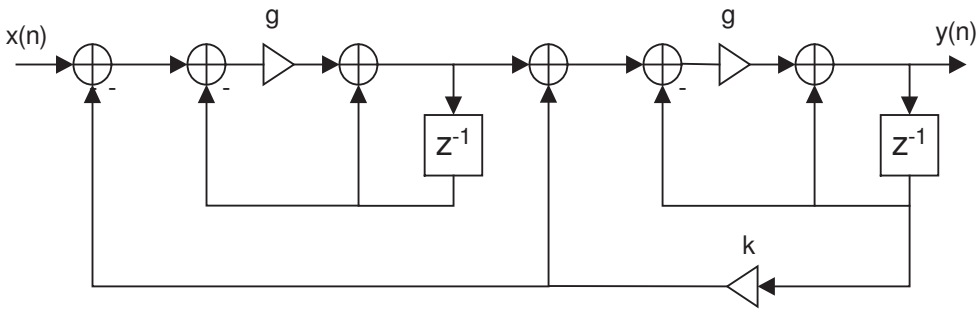

Figure 5.3: Diagram of digital MS-20 filter

algorithms for the DSP.

5.3.1 MS-20 filter

The trivial one-pole discretized MS-20 filter was selected because the low computational requirements were deemed more important than exact response. The structure used is shown in figure 5.3. The implementation uses a slightly different form from that presented in 4.2.3 in which each section is implemented with an ideal integrator and negative feedback. This reduces the number of parameters needed in the innerloop to just two: g for the cutoff coefficient and $resonance$ for resonance amount. The modification allows use of double precision accumulators for the filter memory, decreasing the quantization noise without requiring expensive double precision multiplies.

The coefficient g is linearly interpolated at runtime, avoiding clicks when changing filter cutoff. The values at start and end of frame are linearly interpolated from a pre-calculated table. Resonance is assumed constant for the duration of note. The effect of cutoff on resonance is not compensated in any way. As parameters for each instrument are individually tuned, this nonideality is of little practical concern. Very high resonance values are only used for special effects and some drum sounds.

The accumulators (unit delay inputs) for each section should have at least 32-bit resolution to avoid artifacts when using low cutoff frequencies. Interpolation of g poses a problem for 16-bit DSP when cutoff is slowly varied. It may be best to update g at a lower rate, which also slightly lowers the computational cost. As shown, the filter requires three multiplications and seven additions per sample (taking into account interpolation of g).

5.4 FM

Some instruments require the use of Frequency Modulation synthesis (2.3). As is common in digital implementations, phase modulation is used instead of true frequency modulation. Sine and triangle were chosen as the oscillator waveforms. Depending on the sound, 2 - 4 operators could be used. Feedback FM was not deemed necessary. Phase modulation is relatively robust to word length issues, so no consideration was given to those, apart from using a 32-bit phase accumulator as for the other oscillators. Other than the use of additional modulation term added to the waveform phase, the implementation of sine and triangle waveforms is same as for the regular oscillators.

5.5 Modulation sources

Two types of realtime modulation sources were deemed necessary: Envelopes for controlling the development of note timbre, amplitude and frequency through time and LFOs for producing periodic change in timbre or frequency (vibrato). As the implementation of these depends greatly on the used platform, only some suggestions for implementation are given in this section.

To save computational requirements, modulations are only calculated at the start and end of each frame and the final results interpolated within frame. This is reasonable considering their relatively slow rate of change. The slower update rate of modulations also helps deal with the low bit resolution of the DSP: If modulations were calculated for each sample, the rate of change would be very small for many settings and would approach 1 LSB (Least Significant Bit), which would produce severe errors in timing. By calculating them at a slower rate, referred to here as the control rate F_{ctrl}, the rate of change is faster and numerical errors are less problematic.

5.5.1 Envelope generator

A simple Attack-Decay-Sustain-Release (ADSR) envelope was deemed enough for most sounds. It has a linearly rising attack portion, a decay/sustain portion that exponentially approaches the sustain level and a release portion that exponentially decreases towards zero.

The equations describing these phases are

$$\begin{aligned} y(n) &= y(n-1) + F_{ctrl}/t_a && attack \\ y(n) &= y(n-1) + (1 - e^{-1/(t_d F_{ctrl})})(sustain - y(n-1)) && decay \\ y(n) &= e^{-1/(t_r F_{ctrl})} y(n-1) && release \end{aligned} \tag{5.1}$$

where t_a, t_d and t_r are the attack, decay and release times in seconds respectively, and $sustain$ is the sustain level ($0 <= sustain <= 1$). It can be seen that the equations for the

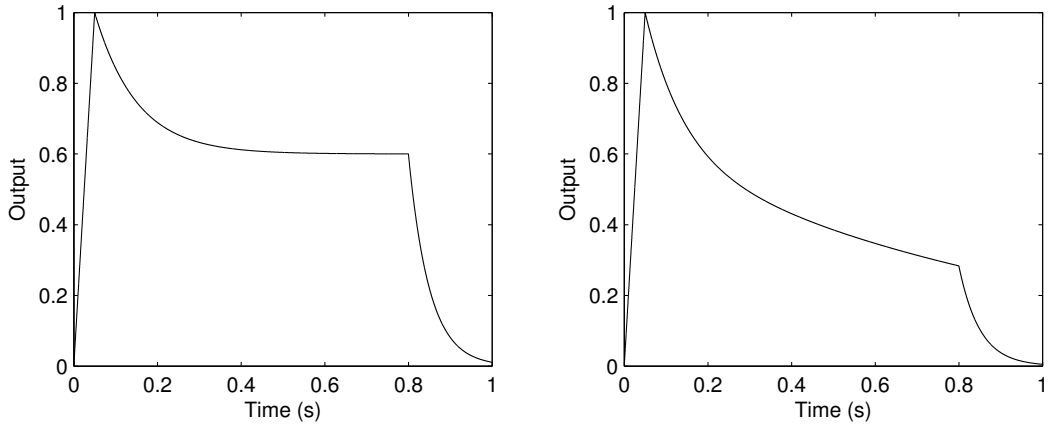

Figure 5.4: ADSR and ADSDR envelopes

decay and release portions are same as for a first order lowpass filter with the input signal being sustain level for the decay portion and zero for the release portion.

Some sounds require more sophisticated control of the attack and decay portion. It was found effective to add a secondary decay parameter for these sounds. This simulates the slow decay of the sound even while the note is held. An easy way to accomplish this is to multiply the regular ADSR envelope with a secondary decay envelope. This secondary decay is started at the same time as the primary envelope reaches decay stage.

The shapes of the envelopes are shown in figure 5.4. For both figures attack time was 0.05 seconds, decay time 0.1 seconds, sustain level 60% and release time 0.05 seconds. The ADSDR envelope has secondary decay of 1.0 second. The note was released at 0.8 seconds in both cases.

5.5.2 Low frequency oscillator

The Low Frequency Oscillator (LFO) is used to slowly change the timbre and / or pitch of an instrument. As such, its speed is usually below audio range and no alias reduction techniques are necessary. The required LFO waveforms are sine, triangle and square. There is also an optional fade-in, to imitate the behaviour of players slowly introducing vibrato.

It may be necessary to use more than 16-bit resolution for the LFO counter due to the long period often used. Other than that, LFO implementation is trivial.

5.6 Percussion and sound effects

Many percussion and effects sounds cannot be efficiently synthesized using existing synthesis methods. Compressed samples [26] are used for those as there was already existing support for ADPCM decoding on the target platform. It was found that using 22 kHz samplerate and duplicating each sample at playback halved the required storage memory without making the sound too muffled or dull. The resulting mirroring of the spectrum to the highest octave worked as a crude but effective form of spectral band replication [15].

A combination of regular and FM oscillators, pseudo noise generator and fixed and varying filters are used for those percussive sounds that can be easily synthesized. Programmable DSP is very useful for this, as each sound can use different combination of synthesis blocks or even ad hoc methods suited only for that particular sound.

Chapter 6

Sound design and examples

This chapter gives a short overview of the sound design process. This is followed by the conventions used in the synthesis blocks and instrument design. Finally, four melodic and one percussion example instrument is presented in detail.

6.1 Sound design

General MIDI (and thus SP-MIDI, too) presents unique challenge for sound design as the standard don't say anything about the timbre or other attributes of an instrument apart from the name. This is exacerbated by the differing opinions manufacturers have had so that an instrument played by two synthesizers from two different manufacturers can sound quite different. As a result existing midi files can have different expectations about the attack / release times of instruments or the effect of note velocity. The sound designer is then designed to cater to a sort of of least common denominator. This sometimes results in having to deliberately make the instruments more 'sterile' sounding to make songs composed on different synthesizers still sound appropriate.

Because of this and the limitations caused by the target architecture, the sound parameters were determined by experimentation and listening. In many cases a realistic sound was not even the goal and instead a sound that gave the impression of the listed instrument and sounded pleasant was opted for. Several articles and books originally aimed at musicians trying to implement "real" instruments on synthesizers were found very useful for the sound design [36, 19]. Also some refences about the physics of real instruments were used for guidance [39].

All sound design was done in MATLAB [27] with the synthesis algorithms implemented as separate functions and then called from the main script of each instrument. This was selected to allow rapid prototyping and experimentation with algorithms. As actual imple-

mentation on the target platform was outside the scope of this work, the choice of MATLAB scripts allowed the people responsible for that to understand and efficiently implement the algorithms and instruments later on.

6.2 Conventions

Some conventions within and between the synthesis blocks were used to ease both the design and the implementation. The example instrument diagrams also follow these conventions.

All sound and modulation generators output signals between -1.0 and +1.0 (for audio and LFO) or 0 and +1.0 (for envelope generators).

All pitch input signals are in octaves, relative to the base pitch, which is middle C (midi note 60, 261.63 Hz). When the note itself is used as input, it is also relative to middle C, so that midi note 60 is equivalent to zero, midi note 72 to +1.0 and midi note 48 to -1.0. This is similar to the volts / octave convention [31] used in analog synthesizers.

On a 16-bit platform this convention allows using for example 4.12 fixed point arithmetic for scaling and summing the modulations with only 0.3 cent error in the result. The result can then be used as input to an interpolated table lookup routine to give the parameter to be used inside a synthesis block. By varying the lookup table, both oscillator frequency (phase accumulator increment) and filter coefficient can be calculated the same way.

The same convention is used for cases when envelope decay times depend on the note pitch. For these cases, the resulting time is

$$T_d = T_{d0} 2^{keytrack(note-60)/12} \tag{6.1}$$

where T_d is the resulting decay time, T_{d0} is the decay time for middle C pitch, $note$ is the midi note and $keytrack$ is the scaling factor.

In the instrument diagrams, audio input is applied to the left side of a block and modulation to the top. For filters only the cutoff can be modulated. For oscillators the choice of modulation destination is denoted next to the modulation signal connection. For modulation of pulse width (PW), the modulation is linear. The same applies for all amplitude modulations.

6.3 Example instruments

6.3.1 String Ensemble 1

Figure 6.1 shows the structure for String Ensemble 1. The instrument uses two oscillators with pulse waveform. The oscillators have slight detuning and are blended in almost equal

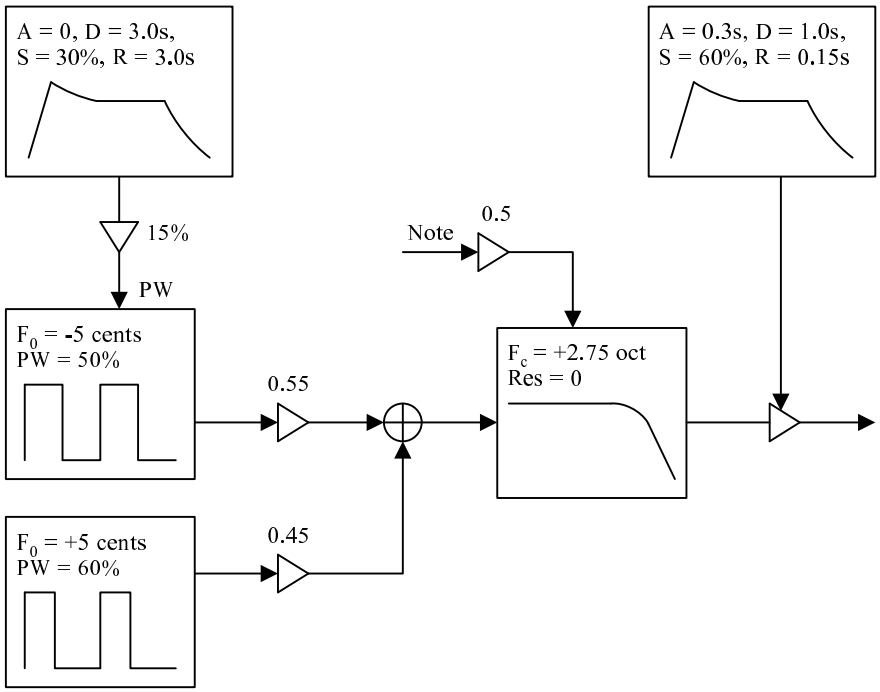

Figure 6.1: String Ensemble 1

parts to produce a pleasant beating effect. The pulse width of the first oscillatore is swept with a very slow decay envelope to add slow variation to the sound.

The filter is set moderately open with no resonance and 50% keyboard tracking is used to smoothly add more high frequencies to higher notes. There is no filter envelope. The amplitude envelope is set for moderately slow attack, slow decay to approximately -4 dB sustain level and moderately slow release.

6.3.2 Electric Guitar (Clean)

Electric Guitar (figure 6.2) uses a sampled single cycle waveform as the basis for the sound. A single note was sampled from a Fender Telecaster guitar and a single cycle (shown in figure 6.3) was separated near the beginning of the note.

The filter envelope was roughly matched to fit the change in timbre, and no resonance is used. The amplitude envelope decay was selected to fit the decay of the sampled note. For added realism, both the filter and the amplitude envelopes' decay times vary with the note pitch so that higher pitches result in faster decay. The release time was selected to roughly simulate damping the string by hand.

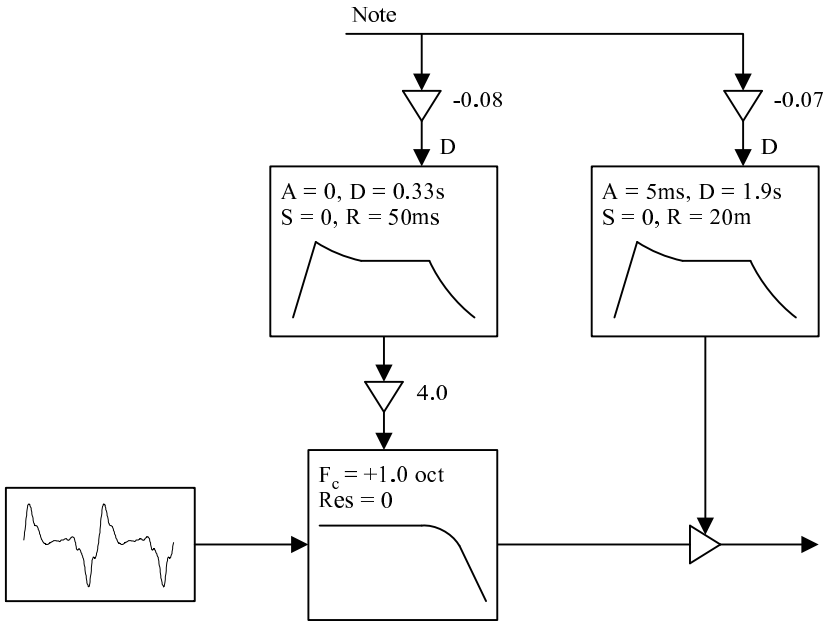

Figure 6.2: Electric Guitar (Clean)

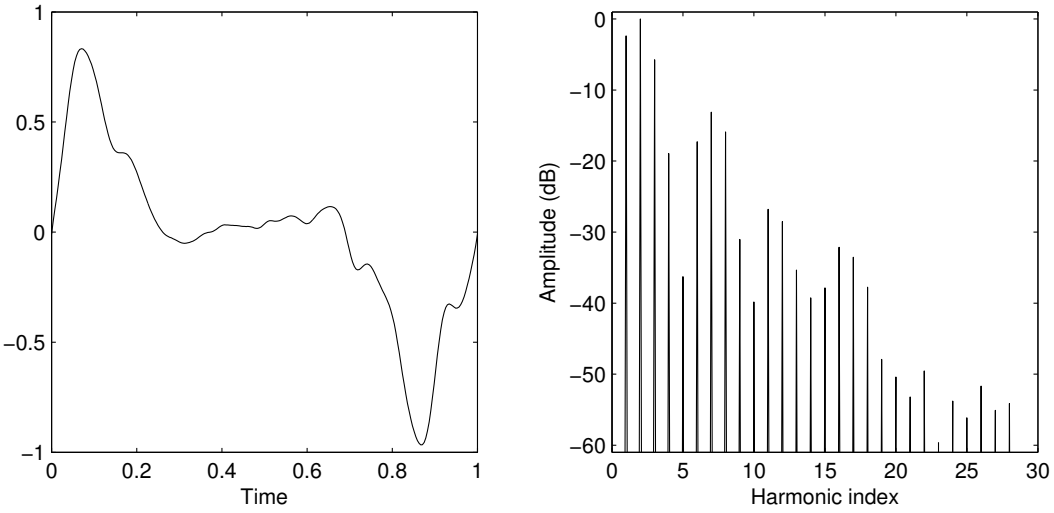

Figure 6.3: A single cycle waveform from Fender Telecaster guitar and its spectrum

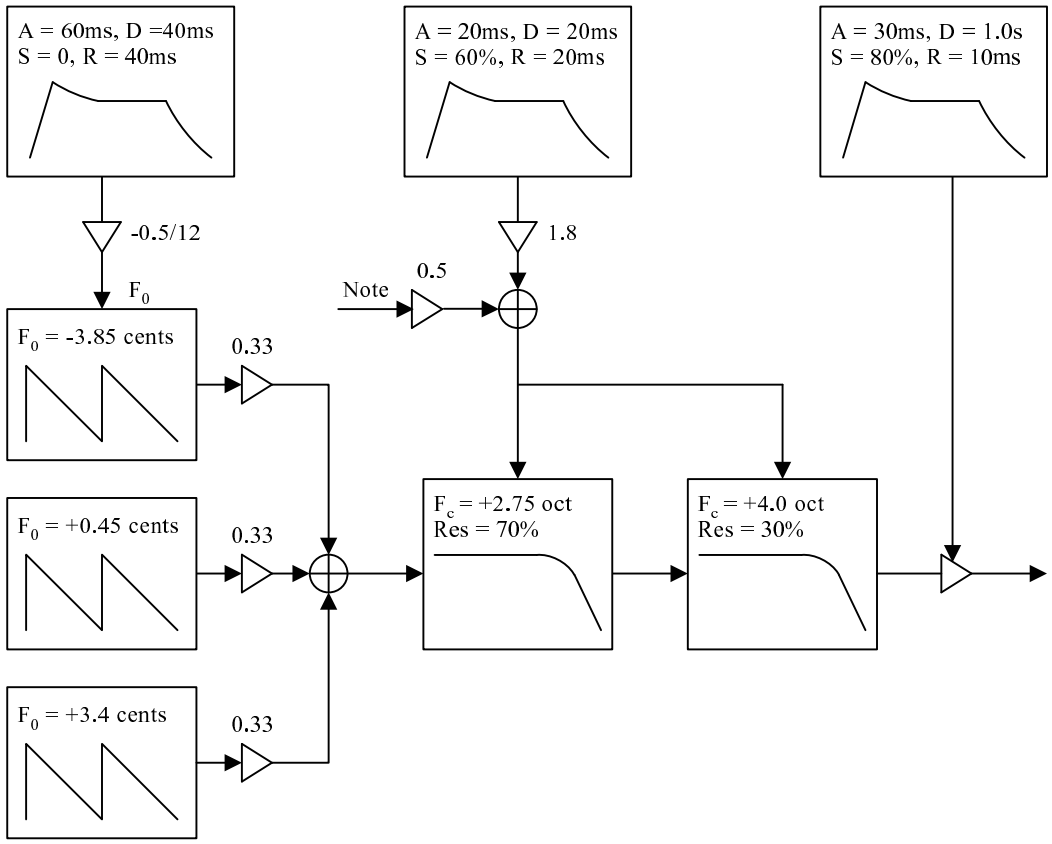

Figure 6.4: Tenor Sax

6.3.3 Tenor Sax

Tenor Sax (figure 6.4) is an example of a more complicated instrument. The three detuned sawtooth oscillators are not strictly correct (saxophone being a single reed instrument [39]), but it was found to produce a more pleasant sound when synthesized. The first oscillator has an initial pitch transient applied from an Attack-Decay envelope during the first 100 - 200 milliseconds of note attack to simulate the inharmonicity present at the initial attack of a sound when the standing wave present in the tube has not yet stabilized [19].

The patch uses two filters in series with each having different cutoff and resonance parameters. Even though fitted only by ear, this gives a rough idea of formants in the sound. The filter has 50% keytrack to add brightness to higher notes. The envelope has both fast attack and decay and has quite large effect on the timbre. The amplitude envelope has fast attack, slow decay to -2 dB sustain level and fast release.

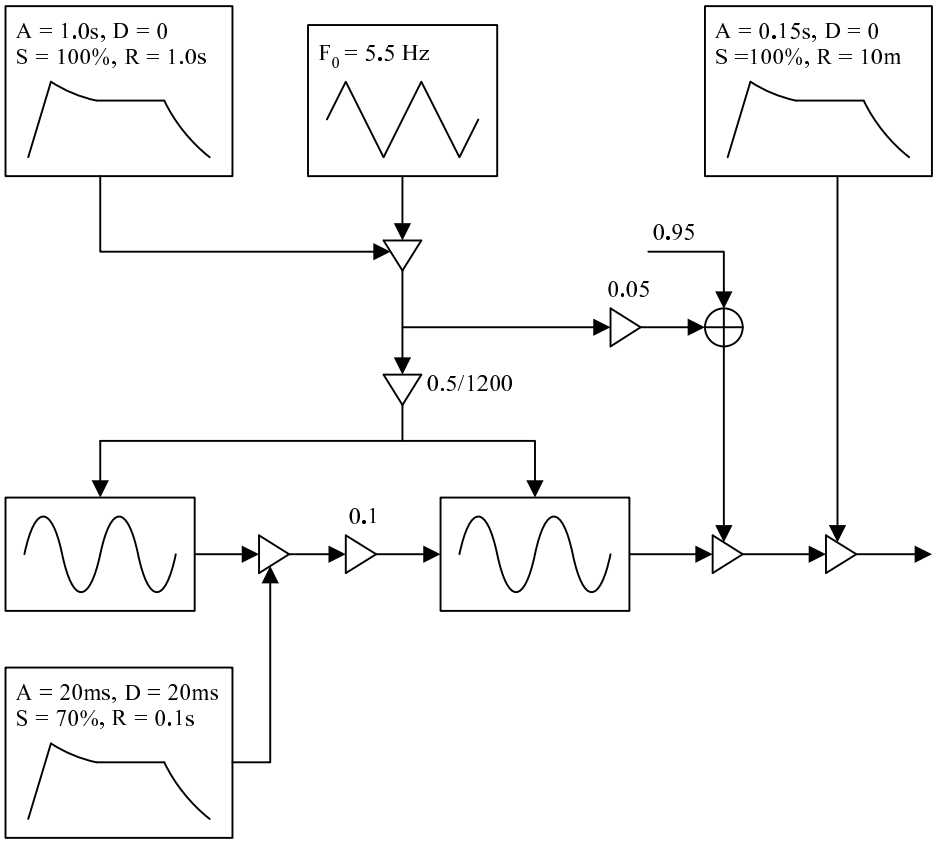

Figure 6.5: Flute

6.3.4 Flute

Flute (figure 6.5 is a typical example of simple FM patch. It uses two oscillators, both with sinusoidal waveform. The modulation index is moderately small. The first oscillator has an associated amplitude envelope with fast attack and decay and 70% sustain level, to give the sound more brightness during the initial attack. The amplitude envelope has moderate attack, full sustain and fast release.

There is a 5.5 Hz LFO that very slightly modulates the amplitude and of the sound and pitch of both oscillators. The LFO has associated envelope that softly introduces the vibrato.

6.3.5 Bass Drum 1

Bass Drum 1 is shown in figure 6.6. It is an example of the routing flexibility a programmable DSP allows. The instrument consists of two main parts: Filtered noise and pitch modulated pseudo- sine.

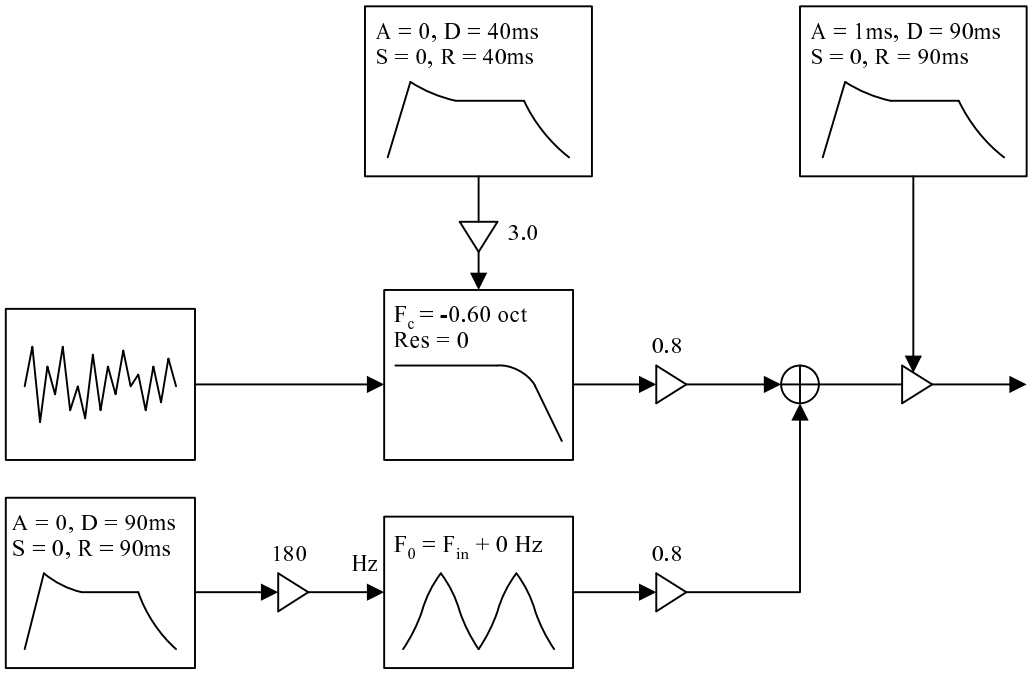

Figure 6.6: Bass Drum 1

The noise part uses regular pseudorandom noise generator (already present on the platform) filtered by a lowpass filter. The filter cutoff is modulated with a fast decay envelope to simulate the noisy attack of a real bass drum.

The pseudo-sine is a hybrid of triangle and sine. It is generated by a triangle oscillator that is scaled to 75% and then used as input to sine lookup table. The result is a sine with sharper peaks. The frequency is swept with a decay envelope, so that

$$f(t) = 180 env(t) \tag{6.2}$$

where $f(t)$ is the pitch of the oscillator in Hz at time t and $env(t)$ is the output of the envelope.

Both parts are summed and then scaled with a common amplitude envelope with very fast attack and moderate decay / release.

Chapter 7

Conclusions and Future Work

In this thesis an was presented of synthesis techniques and sound design suitable for implementing Scalable Polyphony-MIDI instruments on mobile and other devices with limited computation and memory resources. First, an overview of SP-MIDI and a target implementation platform was presented. Then, in chapter 2 several synthesis methods with emphasis on computational efficiency were reviewed, including subtractive synthesis and frequency modulation synthesis, which were the primary synthesis methods used in the work.

Chapter 3 reviewed the waveforms typically used in subtractive synthesis and several techniques for synthesizing them. Special attention was given to the problem of aliasing. Section 3.2.3 presented Differentiated Parabole Wave, an efficient method for generating sawtooth with reduced aliasing. Small corrections to the analysis of established techniques BLIT and (Min)BLEP were presented in sections 3.2.4 and 3.2.5, in addition to an overview of the methods.

An overview of filter structures used in subtractive synthesis was given in chapter 4. In section 4.2.2 several enhancements to the digital Moog lowpass filter were shown. An entirely new filter structure based on the voltage controlled lowpass filter used in the Korg MS-20 analog synthesizer was introduced in section 4.2.3. The novel filter efficiently decouples control of the cutoff frequency and Q for low and medium cutoff frequencies. Even when the effective tuning range is exceeded, Q decreases which guarantees filter stability and coupling can be somewhat corrected by a suitable polynomial or lookup table. Computationally the filter is very efficient, requiring only five multiplications and six additions per sample. The nonlinearity present in the original filter is preserved in the discretization, allowing emulation of analog effects such as amplitude dependent resonance and self-oscillation.

Implementation issues were considered in chapter 5 for selected synthesis algorithms and processing blocks.

Finally, chapter 6 considered sound design and presented several example instruments in detail.

7.1 Future work

Increased use of mobile devices and electronic toys means highly efficient synthesis methods will be relevant in the future. Subtractive synthesis is well suited for implementation on such devices and can be easily coupled with use of samples. The digital MS-20 filter presented is a good candidate for such use because of the high computational efficiency. Further research is needed to find ways to fully decouple cutoff frequency and Q. The MS-20 filter is also relevant in the field of Virtual Analog Synthesis, where the preserved non-linearity is an important advantage.

Subtractive synthesis currently has problems effectively reproducing complex spectra with many peaks and dips such as body resonances of various stringed instruments. Such filters are too complex for a human to design manually, and hence automated methods and tools will be needed. Another problem is modeling the detailed change in spectrum with time, although some attempts have been made in a musical context [41].

Bibliography

[1] Uwe Andresen. A new way in sound synthesis. In *62nd AES Convention*, March 1979.

[2] Daniel Arfib. Digital synthesis of complex spectra by means of multiplication of nonlinear distorted sine waves. *Journal of the Audio Engineering Society*, 27(10):757–768, October 1979.

[3] M. Bellanger. Improved design of long FIR filters using the frequency masking technique. In *Proceedings of International Conference of Acoustics, Speech and Signal Processing*, 1996.

[4] Eli Brandt. Hard sync without aliasing. In *Proceedings of International Computer Music Conference*, 2001.

[5] Robert Bristow-Johnson. The equivalence of various methods of computing biquad coefficients for audio parametric equalizers. In *97th AES Convention*, November 1994.

[6] Robert Bristow-Johnson. Wavetable synthesis 101, a fundamental perspective. In *101st AES Convention*, November 1996.

[7] Marc Le Brun. Digital waveshaping synthesis. *Journal of the Audio Engineering Society*, 27(4):250–266, April 1979.

[8] Hal Chamberlin. *Musical Applications of Microprocessors*. Hayden Books, 1987.

[9] Amar Chaudhary. Band-limited simulation of analog synthesizer modules by additive synthesis. In *105th AES Convention*, September 1998.

[10] John M. Chowning. The synthesis of complex audio spectra by means of frequency modulation. *Journal of the Audio Engineering Society*, 21(7):526–534, September 1973.

[11] Curtis Electromusic Specialties. *CEM 3328 Four Pole Low-Pass VCF datasheet*, 1984.

[12] Jon Dattorro. Effect design part 1: Reverberator and other filters. *Journal of the Audio Engineering Society*, 45(9):660–684, September 1997.

[13] Jon Dattorro. Effect design part 3 oscillators: Sinusoidal and pseudonoise. *Journal of the Audio Engineering Society*, 50(3):115–146, March 2002.

[14] Laurent de Soras. The quest for the perfect resampler. http://ldesoras.free.fr, June 2003.

[15] Martin Dietz, Lars Liljeryd, Kristofer Kjörling, and Oliver Kunz. Spectral band replication, a novel approach in audio coding. In *112th AES Convention*, May 2002.

[16] Yinong Ding and Dave Rossum. Filter morphing of parametric equalizers and shelving filters for audio signal processing. *Journal of the Audio Engineering Society*, 43(10):821–826, October 1995.

[17] Pierre Dutilleux and Udo Zölzer. *DAFX - Digital Audio Effects*, chapter 2. John Wiley & Sons, 2002.

[18] Andrew Horner, James Beauchamp, and Lippold Haken. Machine tongues XVI: Genetic algorithms and their application to FM matching synthesis. *Computer Music Journal*, 17(1):17–29, 1993.

[19] Alex Noyes Howard Massey and Daniel Shklair. *A Synthesist's Guide to Acoustic Instruments*. Amsco Publications, 1987.

[20] Antti Huovilainen. Non-linear digital implementation of the Moog ladder filter. In *Proceedings of the 7th International Conference on Digital Audio Effects (DAFx-04)*, 2004.

[21] Korg Corporation. *Korg MS-20 KLM-127 board schematic (version 2)*, accessed April 30th, 2010.

[22] Yuyo Lai, Shyh-Kang Jeng, Der-Tzung Liu, and Yo-Chung Liu. Automated optimization of parameters for FM sound synthesis with genetic algorithms. In *Proceedings of International Workshop on Computer Music and Audio Technology*, 2006.

[23] John Lane, Dan Hoory, Ed Martinez, and Patty Wang. Modeling analog synthesis with DSPs. In *Proceedings of Internation Computer Music Conference*, 1997.

[24] John E. Lane. Pitch detection using a tunable IIR filter. *Computer Music Journal*, 14(3):46–59, 1990.

[25] Victor Lazzarini and Joseph Timoney. New perspectives on distortion synthesis for virtual analog oscillators. *Computer Music Journal*, 34(1):28–40, 2010.

[26] Robert C. Maher. Wavetable synthesis strategies for mobile devices. *Journal of the Audio Engineering Society*, 53(3):205–212, March 2005.

[27] Mathworks. MATLAB. http://www.mathworks.com/products/matlab/, accessed April 30th, 2010.

[28] The MIDI Manufacturers Association. http://www.midi.org/, accessed April 30th, 2010.

[29] The MIDI Manufacturers Association. *Scalable Polyphony MIDI Specification*, May 2002.

[30] The MIDI Manufacturers Association. *DLS Level 2 Specification*, April 2006.

[31] Robert A. Moog. A voltage-controlled low-pass high-pass filter for audio signal processing. In *17th AES Convention*, 1965.

[32] F. Richard Moore. Table lookup noise for sinusoidal digital oscillators. In Curtis Roads and John Strawn, editors, *Foundations of Computer Music*, pages 326–334. MIT Press, 1985.

[33] Olli Niemitalo. Polynomial interpolators for high-quality resampling of oversampled audio. http://www.iki.fi/o/dsp/deip.pdf, August 2001.

[34] W. H. Press. *Numerical Recipes, The Art of Scientific Computing*. Cambridge University Press, 1989.

[35] L.R. Rabiner, J.H. MCClellan, and T.W. Parks. FIR digital filter design techniques using weighted Chebyshev approximations. In *Proceedings of IEEE*, 1975.

[36] Gordon Reid. Synth secrets column. *Sound on Sound*, 1999-2004. http://www.soundonsound.com/sos/allsynthsecrets.htm.

[37] Curtis Roads. *The Computer Music Tutorial*, chapter 5. MIT Press, 1996.

[38] X. Rodet and P. Depalle. Spectral envelopes and inverse FFT synthesis. In *93rd AES Convention*, October 1992.

[39] Thomas D. Rossing, Richard F. Moore, and Paul A. Wheeler. *The Science of Sound*. Addison Wesley, 3rd edition, 2001.

[40] Dave Rossum. The 'Armadillo' coefficient encoding scheme for digital audio filters. In *Proceedings of IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, 1991.

[41] Martin Russ. E-Mu Morpheus synth module review. *Sound on Sound*, August 1994.

[42] Martin Russ. Yamaha VL1 review. *Sound on Sound*, July 1994.

[43] Andrew Simper. Personal communication, 2006.

[44] Julius O. Smith. Physical modeling using digital waveguides. *Computer Music Journal*, 16(4):74–91, 1992.

[45] Julius O. Smith. *Introduction to Digital Filters with Audio Applications*. http://ccrma.stanford.edu/~jos/filters/, online book, accessed April 30th, 2010.

[46] Julius O. Smith. *Physical Audio Signal Processing for Virtual Musical Instrument and Audio Effects*. http://ccrma.stanford.edu/~jos/pasp, online book, accessed April 30th, 2010.

[47] Tim Stilson and Julius O. Smith. Alias-free digital synthesis of classic analog waveforms. In *Proceedings of International Computer Music Conference*, 1996.

[48] Tim Stilson and Julius O. Smith. Analyzing the Moog VCF with considerations for digital implementation. In *Proceedings of International Computer Music Conference*, 1996.

[49] Matti Karjalainen Timo I. Laakso, Vesa Välimäki and Unto K. Laine. Splitting the unit delay - tools for fractional delay filter design. *IEEE Signal Processing Magazine*, 13(1):30–60, January 1996.

[50] Tero Tolonen, Vesa Välimäki, and Matti Karjalainen. Evaluation of modern sound synthesis methods. Technical report, Helsinki University of Technology, Espoo, Finland, 1998.

[51] Norio Tomisawa. Tone production method for an electronic musical instrument. US Patent 4,249,447, 1981.

[52] VLSI Solution Oy. *VS1003b MP3/WMA Audio Codec datasheet*, February 2009.

[53] Vesa Välimäki. Discrete-time synthesis of the sawtooth waveform with reduced aliasing. *IEEE Signal Processing Letters*, 12(3):214–217, March 2005.

[54] Vesa Välimäki, Jyri Huopaniemi, Matti Karjalainen, and Jánosy Zoltán. Physical modeling of plucked string instrument with application to real-time sound synthesis. *Journal of the Audio Engineering Society*, 44(5):331–353, May 1996.

[55] Vesa Välimäki and Antti Huovilainen. Oscillator and filter algorithms for virtual analog synthesis. *Computer Music Journal*, 30(2):19–31, 2006.

[56] Vesa Välimäki and Antti Huovilainen. Antialiasing oscillators in subtractive synthesis. *IEEE Signal Processing Magazine*, 24(2):116–125, March 2007.

[57] Vesa Välimäki, Jyri Pakarinen, Cumhur Erkut, and Matti Karjalainen. Discrete-time modelling of musical instruments. *Reports on Progress in Physics*, 69(1):1–78, 2006.

[58] Yamaha Corporation. *DX7 Owner's Manual*, 1983.

[59] David T. Yeh, Jonathan S. Abel, and Julius O. Smith. Simplified, physically-informed models of distortion and overdrive guitar effect pedals. In *Proceedings of the 10th International Conference on Digital Audio Effects (DAFx-07)*, 2007.