Audio Engineering Society

Convention Paper 5937

Presented at the 115th Convention
2003 October 10–13 New York, New York

Compression and Decompression of Wavetable Synthesis Data

Robert C. Maher

Dept. of Electrical and Computer Engineering, Montana State University, Bozeman, MT 59717-3780 USA
email: rob.maher@montana.edu

ABSTRACT

Table look-up (or wavetable) synthesis methods have been widely used for many years in music synthesizers. Recently wavetable music synthesis has received new interest in the context of mobile applications such as personalized ring tones and pager notification signals. The limited amount of storage and processing available in such mobile devices makes the use of compressed wavetable data desirable. In this paper several issues related to wavetable compression are described and an efficient compression/decompression method is proposed for reducing the size of wavetable data while maintaining loop continuity and timbral quality.

1. INTRODUCTION

Digital table-lookup synthesis has been widely used in music synthesis applications [1, 2, 3]. In its most basic form, a single cycle of the desired waveform is digitized and then simply played back repeatedly (looped) to create a sustained signal. In order to create musically interesting synthesized signals it is necessary to have a set of stored waveform tables that correspond to sounds from a variety of musical instruments, or from different pitch and amplitude ranges of the same instrument. The size of the set of lookup tables can become quite large if a wide range of timbres at a high level of quality is desired, and the complexity of selecting, manipulating, and possibly cross-fading various tables in real time can be significant. Nonetheless, the table lookup technique has been employed in many commercial products and computer music synthesis systems.

Recently there has been renewed interest in the use of stored waveform music synthesis for personalized ring and page tones in mobile telephony [4]. The extremely limited amount of storage, transmission bandwidth, and available computation, not to mention the prevalent cost-reduction pressure of the competitive marketplace, indicates the need for a simple and efficient means to reduce the size of stored wavetables while still allowing low complexity decompression.

In order to achieve a reduction in the storage required for the wavetable data, the number of bits used to represent each sample must be reduced. It is, of course, desirable to perform the bit rate reduction losslessly so that the original wavetable can be recovered exactly from the compressed data. However, to achieve a useful data compression factor and minimal computation it is helpful to consider a quantized (lossy) representation, yet still meet the loop continuity and Nyquist constraints.

The method proposed in this paper is to use a nonuniformly quantized differential pulse-code modulation (DPCM) procedure. The difference between the current wavetable sample and the next sample is calculated then quantized with a nonuniform characteristic, such as A-law, μ-law, or logarithmic [5]. This allows more resolution for small signal details and relatively less accuracy for large signal changes. The output is then recovered by decoding the sample differences and summing them to obtain the output waveform.

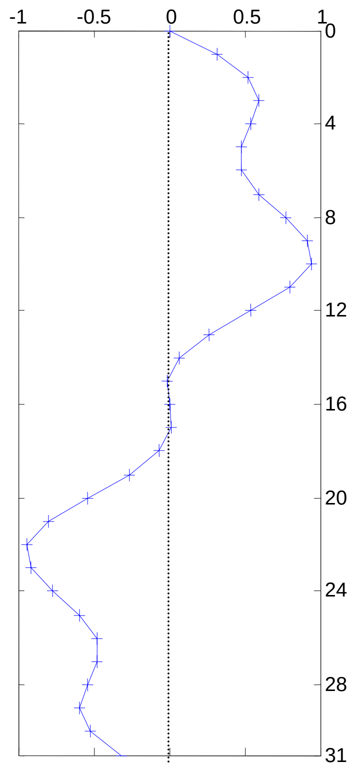
Due to the quantization process, this procedure is a many-to-one mapping and is therefore lossy. As with any lossy coding operation the discrepancy between the original signal and the reconstructed signal may or may not be objectionable depending upon the nature of the signals and the application context. However, the use of lossy coding for looped wavetables poses several peculiar problems. First, the reconstruction obtained from the quantized differential data may not allow a "perfect" loop (amplitude match at the end points), which may cause an audible click or DC buildup as the waveform is cycled. Second, the introduction of quantization errors into the wavetable data occurs *after* the antialiasing (or resampling) filters that are used by the sound designer in the looping process, and the resulting error signal is unlikely to be properly bandlimited, resulting in unintended aliasing in the wavetable signal. Finally, the use of common statistical procedures such as dither and noise feedback to ameliorate the audible effects of the quantizer are not appropriate for wavetable coding, since the added signals are "frozen" into the stored wavetable and looped, rather than being statistically independent as in ordinary digital audio processing.

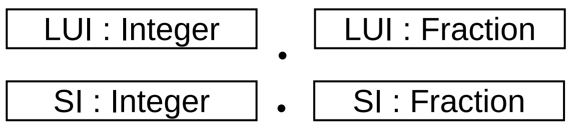
The remaining sections of this paper are organized as follows. First, a summary of wavetable principles and differential encoding is given. Next, the proposed low-complexity encode/decode scheme for wavetable data is presented. Finally, several variations of the proposed technique are discussed.

2. WAVETABLE SYNTHESIS

A wavetable or stored-waveform synthesizer operates by repeatedly sending an array of waveform samples—the *wavetable*—through a digital-to-analog (D/A) converter. The basic schema of a wavetable synthesizer for N=32 is shown in Figure 1.

Index	Sample
0	0.000
1	0.313
2	0.525
3	0.591
4	0.544
5	0.476
6	0.480
7	0.591
8	0.769
9	0.918
10	0.941
11	0.803
12	0.544
13	0.264
14	0.064
15	-0.013
16	0.000
17	0.013
18	-0.064
19	-0.264
20	-0.544
21	-0.803
22	-0.941
23	-0.918
24	-0.769
25	-0.591
26	-0.480
27	-0.476
28	-0.544
29	-0.591
30	-0.525
31	-0.313





Current Index = LUI : Integer
Next LUI = (LUI + SI) modulo N

Figure 1: Example wavetable of length N=32. Look-Up Index (LUI) and Sample Increment (SI) are rational numbers used to set the table repetition frequency.

It is generally necessary to be able to synthesize a periodic waveform of arbitrary length, so the wavetable is often filled with one or more cycles of a periodic waveform which is then repeatedly cycled, or *looped*, as long as necessary. If the array of samples is of length N and the D/A converter is clocked at a fixed rate f_s samples per second, it requires N/f_s seconds to read every sample from the table. This corresponds to a waveform frequency of f_s/N Hz, assuming the table contains just one waveform cycle.

2.1. Resampling and Interpolation

For music synthesis it is usually necessary to produce waveforms with differing and perhaps time-varying fundamental frequencies, and certainly frequencies that are not integer divisors (f_s/N) of the sample frequency [6]. This requires sample rate conversion of the wavetable data, meaning that waveform samples in between the given samples of the wavetable will need to be interpolated. One common way to keep track of the interpolation is to define a *sample increment* value (integer and fraction) that indicates the required hop through the table in terms of the original f_s and table length N.

$$SI = N \cdot (f_{desired}/f_s)$$

The location within the table where interpolation is to occur is the *lookup index* (*LUI*), which is computed recursively by

$$LUI[n] = \{\, LUI[n\text{-}1] + SI \,\} \bmod N$$

where "mod N" indicates a mathematical integer modulus operation. This simply implies that *LUI* wraps around the end of the wavetable. Since *LUI* is in general a real number and the stored wavetable samples correspond to integer indices, the wavetable value at *LUI* can be interpolated in a number of ways, such as rounding or truncating *LUI* to the nearest integer, using linear interpolation, or using some higher-order interpolation function. For example, linear interpolation involves the two wavetable samples adjacent to *LUI*, namely $w\{(\text{int})\,LUI[n]\}$ and $w\{(1+(\text{int})\,LUI[n])\bmod N\}$, and can be accomplished by:

$$\begin{aligned} w_{\text{lin\_interp}}\{LUI[n]\} =& \\ & w\{(\text{int})LUI[n]\} + \\ & [\quad w\{(1+(\text{int})LUI[n]) \bmod N\} \\ & \quad - w\{(\text{int})LUI[n]\} \quad] \cdot (\text{fract})LUI[n] \end{aligned}$$

where (int) and (fract) indicate the integer part and fractional part, respectively, of *LUI*.

Higher-order interpolation can provide a closer approximation to bandlimited interpolation of the wavetable, at the expense of additional computation: more than the two adjacent wavetable samples are required for a higher-order structure.

There are two important implications of the resampling/interpolation procedure used in the wavetable synthesizer. First, the resampling process condenses or expands the spectrum of the signal stored in the wavetable—analogous to a tape being played faster or slower—and this can cause timbral changes (formant shifts) that are usually undesirable. Second, the resampling process may introduce aliasing if the interpolation is not strictly bandlimited or if the wavetable is critically sampled to begin with. Since it is advantageous to minimize the size (length) of the wavetable for practical reasons, there is a tradeoff between using a longer wavetable with oversampled data to allow easier resampling, or using a shorter wavetable with critically sampled data to conserve storage space.

2.2. Other Considerations

In addition to the cyclical wavetable described above, it is common in practice to provide additional features for enhancing the flexibility and sound quality of the wavetable procedure. One important feature is to allow an attack waveform to be spliced in front of the cyclical wavetable, as depicted in Figure 2.

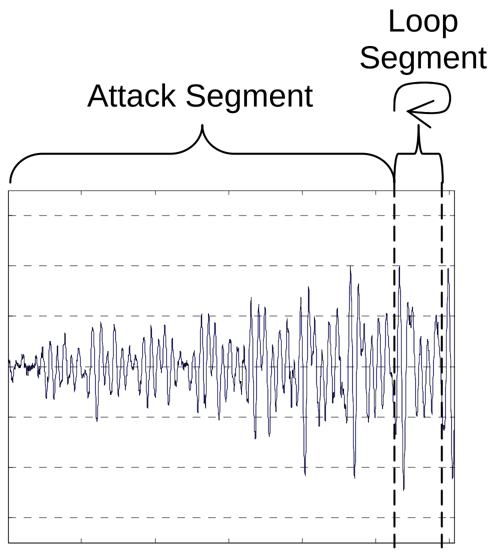


Figure 2: Example of a wavetable with a one-shot attack portion and a sustaining loop segment.

The attack portion is played once at the onset of the musical note, while the cyclical wavetable is used to sustain the length of the note as necessary. The use of a one-shot attack segment allows for more realistic and/or musically interesting timbres.

Another common feature is to provide an amplitude envelope function to create a smoothly varying onset and release of the wavetable signal. More sophisticated wavetable synthesizers may include frequency selective filters, provision for frequency vibrato and amplitude tremolo, layering of multiple wavetables, elaborate control interfaces, and so forth.

A critical requirement of a successful wavetable synthesis system is to have carefully prepared wavetable data. The process of creating the wavetable source material and ensuring that the loop points are minimally audible must be performed by a skilled sound designer.

3. THE WAVETABLE DATA COMPRESSION PROBLEM

In many applications the size of the wavetable data for a synthesizer can be problematic. A set of wavetables for a synthesizer supporting the 175 timbres required by the General MIDI Level 1 (GM1) specification [7] is typically at least 1 MB, and for high quality applications it may be 4 MB, 12 MB, or even more. Tens of megabytes of storage may not be an issue in the context of a modern personal computer, but for small, inexpensive, and mobile devices such as cell phones or personal digital assistants (PDAs) even an 8 MB wavetable data set may be more than an order of magnitude too large for practical use.

The synthesizer designer has several options for decreasing the size of the wavetables, but generally this involves a corresponding loss of signal quality. Some common strategies to reduce the wavetable storage size are the following.

- The length of the one-shot attack sections can be compromised or eliminated.
- The same wavetable data can be re-used for many different timbres, either without modification or with real time changes to disguise the data using amplitude envelopes, filter settings, etc.
- The synthesizer can be run at a lower sample rate, meaning fewer stored samples per cycle of the waveforms and a lower audio bandwidth.
- The number of bits used to represent each sample in the wavetables can be reduced, such as going from 16-bits per sample to 12-bits or 8-bits per sample.
- The wavetable data can be encoded using a lossless or lossy method.

The latter strategy—a simple wavetable data compression/decompression method—can be quite helpful to reduce the required data storage size while maintaining as much signal quality as possible. The proposed low-complexity wavetable coding procedure is described next.

4. DIFFERENTIAL REPRESENTATION

Many musically useful signals have a lowpass characteristic: the spectral level is a declining function of frequency. One feature of this type of signal is a broad autocorrelation function due to the relatively slow amplitude variations of the lowpass waveform. The high sample-to-sample correlation means that the adjacent sample difference signal $d[n] = x[n] – x[n-1]$ will have a smaller variance than the signal $x[n]$ itself. In other words, the signal is at least somewhat predictable from one sample to the next. We can exploit this *redundancy* to reduce the required bit rate, and thereby represent the signal with fewer total bits [5].

If we accumulate (sum) the sequence $d[n]$ we can compute the reconstructed signal $x_r[n] = d[n] + x_r[n-1]$. If $d[n]$ is received without error or loss, then $x_r[n] = x[n]$, as depicted in Figure 3.

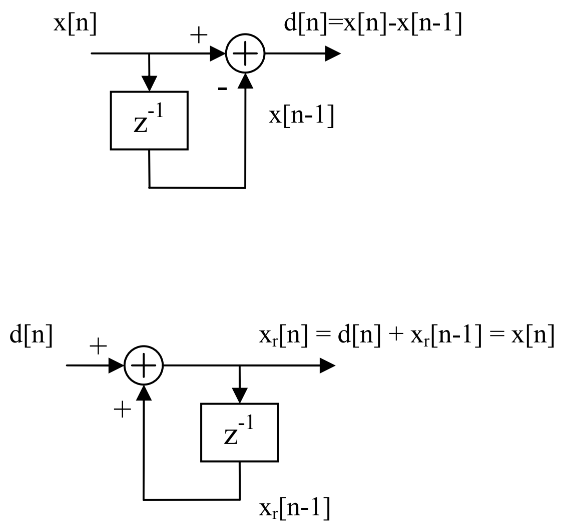
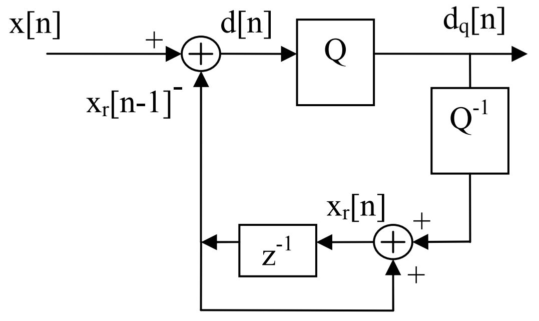


Figure 3: Simple differential encoder and decoder without quantization. The z^{-1} indicates a single sample delay.

Although some data reduction is possible due to the difference signal d[n] being of smaller variance (or average amplitude) than x[n], it is typically necessary to reduce the number of bits per sample still further to obtain a useful amount of compression (e.g., 30-50% reduction). The usual data reduction approaches for a differential waveform coder are to attempt further redundancy reduction by improving the predictor, and to quantize the difference signal more coarsely by allowing lossy coding ($x_r[n] \neq x[n]$). For the wavetable data compression task it is necessary to keep the decompression computation as quick and simple as possible while still obtaining satisfactory signal quality. The proposed compression method therefore uses the coarse quantization approach.

Since the difference signal d[n] is to be quantized more coarsely—becoming $d_q[n]$—the reconstructed signal $x_r[n] = Q^{-1}\{d_q[n]\} + x_r[n-1]$ is based only on the quantized difference values and therefore the coarse quantization of one step in the difference sequence directly influences the reconstructed signal thereafter. Rather than the simple differential encoder of Figure 3, a better approach to help track the discrepancy between x[n] and $x_r[n]$ is to quantize the difference between x[n] and $x_r[n-1]$ and use this as the encoded stream, as shown in Figure 4.



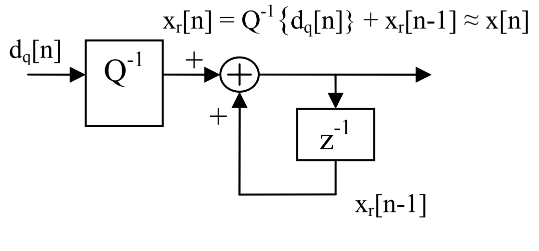


Figure 4: Differential encoder and decoder with a quantizer (Q). Q reduces the number of bits required to represent d[n], causing a discrepancy between x[n] and the recovered signal $x_r[n]$.

Thus, the encoder and the decoder will track each other assuming there are no errors in the storage and retrieval process.

It should be noted that if extra computation is available it may indeed be possible to improve signal quality by using a higher-order predictor in the differential quantization feedback path.

It is clearly an advantage to minimize the audible difference between the original wavetable samples x[n] and the reconstructed wavetable $x_r[n]$. In the proposed wavetable compression method a non-uniform quantizer Q is used to obtain essentially lossless reconstruction when d[n] is small, while allowing some discrepancy if d[n] is large. A variety of non-uniform quantizers can be selected. Common choices include A-law, μ-law, or logarithmic.

4.1. Wavetable Compression Details

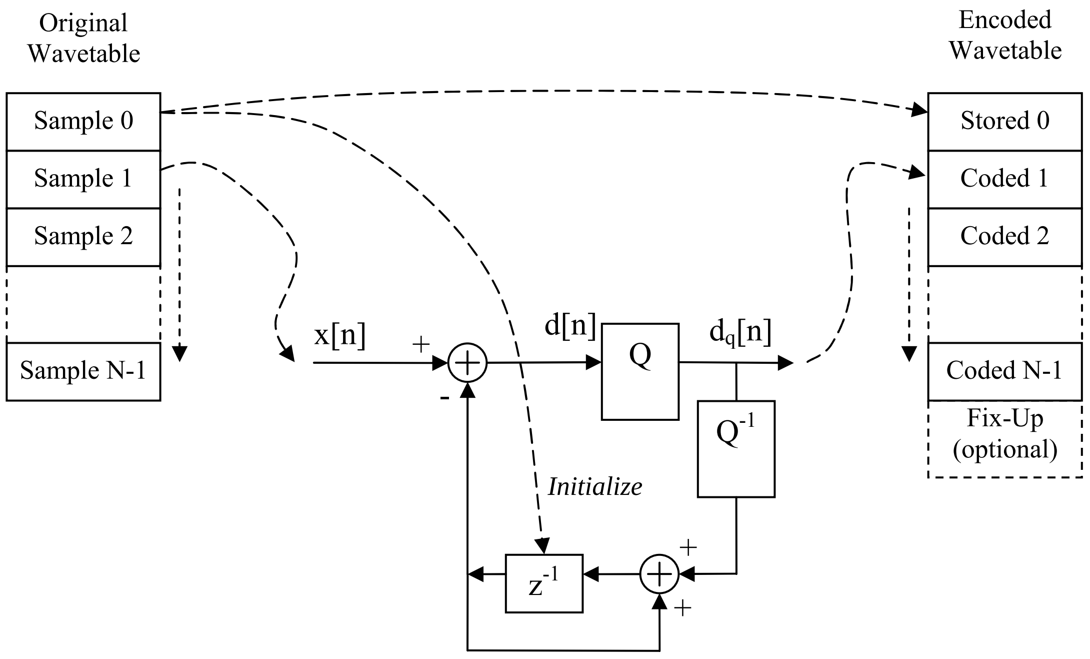
The use of a lossy differential representation introduces distortion. Conventional audio coders often treat the coding distortion as an additive uncorrelated white noise process on a sample by sample basis [5]. However, for wavetable compression the coding distortion is embedded in the decoded waveform and is therefore looped over and over along with the desired signal itself, so simple additive noise models are inappropriate

The distortion components will also introduce aliasing when the wavetable is resampled by the synthesizer to change the musical pitch. Moreover, the correlation between the distortion components and the desired waveform can result in audible noise modulation when the sample increment is changed for vibrato or other musical pitch effects. It is also important to treat the end-of-loop condition carefully so that no discontinuity is introduced at the loop boundary.

These special considerations require extra care during the sound design phase. In particular, the sound designer must be aware of the audible characteristics introduced by the differential encoding process and choose wavetables that reflect the strengths and weaknesses of the encoding system.

5. IMPLEMENTATION FRAMEWORK

Creating a wavetable sample set requires several steps. One common approach is to obtain recordings of the desired musical timbre, select appropriate attack and sustain (loop) segments, and apply sample rate conversion and amplitude quantization to



$$d_q[n] = Q\{d[n]\} = \mathrm{sgn}(d[n]) \cdot \mathrm{floor}\{0.5 + K \cdot \log_{10}(|d[n]|)\}$$

$$Q^{-1}\{d_q[n]\} = \mathrm{sgn}(d_q[n]) \cdot 10^{|d_q[n]|/K}$$

Figure 5: Wavetable encoding process using a log-based non-linear quantizer. The coded values have fewer bits of precision than the original samples. *K* is a scaling constant chosen to provide the desired encoded signal resolution.

achieve a reasonable compromise between wavetable length, word size, and susceptibility to aliasing.

5.1. The Encoding Process

The encoding process is expected to be done once by a skilled sound designer “at the factory” rather than by the user in the field. Thus, it is feasible to expend some effort to adjust the precise sampling rate and amplitude to minimize the encoding error and distortion introduced by the lossy differential wavetable representation. The sound designer must be prepared to evaluate the intermediate results and adjust the waveform parameters as necessary.

An example of the wavetable encoding process is shown in Figure 5. First, the initial (unencoded) value of the wavetable is stored. Next, the first difference is calculated using the method of Figure 4 and a non-uniform quantizer, Q. The quantizer can be implemented in a variety of forms. For this example an explicit logarithmic formula is shown, although a lookup table could also be used. The sequence of quantized differences is determined and the differential (encoded) values are stored. At the end of the encoded wavetable it may be helpful to calculate the exact (unencoded) difference between the last accumulated wavetable sample and the first sample of the loop (see Method 3 below). If a higher-order predictor is used in the encoder, the initial values of the internal predictor coefficients may also be stored for use by the decoder.

5.2. The Decoding Process

While the encoding process can involve iterative procedures and the services of a skilled sound designer, the decoding process must be extremely efficient and sufficiently fast to allow real time operation. The precise nature of the decoder may vary from one product platform to another, but it is generally necessary to keep the decoding complexity of the same order as the wavetable synthesis process itself. This implies that a lookup table in the decoder might be preferable to an algorithm to compute exponentials or transcendental functions.

Three possible decoding scenarios are considered next.

5.2.1. Method 1: Full decode buffer.

The first scenario to be considered for wavetable reconstruction is the use of a decode buffer holding the entire decoded wavetable. The encoded data is read from storage and the entire wavetable loop is decoded into a buffer, or *cache*, for the synthesizer. This method requires enough buffer storage for the entire wavetable loop, but only the active wavetable data needs to be present. The decode operation occurs only when a new musical note calls for a different wavetable.

The advantage of the full decode buffer is that it may allow the synthesizer functions to be simplified, since the decoding operation is accomplished outside of the sample-by-sample computation of the synthesizer. It is also possible for multiple concurrent synthesized voices to share the same cached wavetable. The primary disadvantages of this approach are the need to delay while the entire wavetable is decoded, and the obvious need for substantial buffer memory for the active wavetable cache.

5.2.2. Method 2: Accumulator with loop reset

In this scenario the synthesizer fetches and decodes only sufficient wavetable data “on-the-fly” to produce the current block of output samples. An accumulator variable is loaded with the initial wavetable value, then each decoded wavetable sample is obtained by adding (accumulating) the sequence of differential values over the required *LUI* (look-up index) range for the current block. Once the current wavetable block is synthesized the last value of the accumulator is saved for use in the next block. If *LUI* traverses the end of the wavetable data, the wraparound condition is detected, the accumulator is loaded again with the initial wavetable value, and the process continues.

The accumulator method has the advantage that minimal buffer storage is required. It also allows the synthesis process to begin more rapidly than Method 1, since only the required wavetable samples need to be decoded. The accumulator method does have the disadvantage that the fetch and decoding operations must occur over and over as long as the synthesizer sustains the note, thereby increasing the average computational cost per output sample.

A more subtle disadvantage is that the differential encode/decode implies that every encoded wavetable element must be fetched and accumulated even if the intervening samples are not actually needed by the synthesizer. For example, if the sample increment (*SI*) is greater than unity and only simple linear interpolation is performed, certain wavetable samples will be skipped as the lookup index hops through the wavetable, but the skipped samples must still be computed due to the sequential differential representation of the wavetable.

5.2.3. Method 3: Accumulator with loop fix-up

The third scenario is similar to Method 2, except a special “fix-up” value is used rather than resetting the decode accumulator at the end of the loop. The fix-up value allows a perfect loop accumulation by holding the unencoded difference between the accumulated differential values at the end of the wavetable and the value at the start of the loop. The fix-up is pre-calculated during the encoding process and stored at the end of the wavetable. By reserving one of the code words to indicate that the fix-up value is to be fetched, the decode/synthesis process can be implemented with a symmetrical structure that does not require explicit end-of-loop calculations. This approach is particularly suited to situations in which the sample fetch, decoding, and interpolation processes are performed by special-purpose hardware.

6. CONCLUDING COMMENTS

The differential wavetable encoding method can achieve data compression factors of 30-50% with acceptable quality. This corresponds to representing the original 16-bit audio samples with only 8-bit nonlinear differential values, while still achieving roughly 12-bit quality.

6.1. Distortion Minimization

The nonlinear quantizer (e.g., log, μ-law, etc.) may make the encoder quite sensitive to the amplitude of the original wavetable data. It has been found that performing a systematic search for the minimum level of distortion can be helpful. During the sound design process the wavetable data are automatically encoded repeatedly with different amplitude scalings and the RMS discrepancy between the decoded wavetable and the original data is calculated. The amplitude scaling that results in the lowest distortion is then stored for use by the synthesizer. The nonlinear nature of the differential quantization process prevents an easy analytical solution, but the availability of fast computers makes the iterative optimization very feasible. The use of an objective perceptual quality measurement might also be helpful for this purpose.

6.2. Log Encoding to Reduce Multiplies

In typical wavetable synthesis applications the samples recovered from the wavetable are interpolated using a multiplicative weighting, and then further multiplied by a time-varying amplitude envelope to simulate the attack and release characteristics of the desired timbre. The multiplication operations may be costly in a custom silicon implementation due to the size and complexity of the multiplier hardware. However, by using a true logarithmic encoding of the wavetable differential values and representing the amplitude envelope in logarithmic form, it has been shown that the interpolation and amplitude scaling can be performed by *adding* the logarithms rather than requiring explicit multiplies [8]. An anti-log lookup table in the decoder can then be used to reconstruct the output waveform.

6.3. Model-based compression

Another encoding alternative consists of a linear prediction model. In this case the wavetable is considered to be a portion of the impulse response of a recursive digital filter. The encoding process involves determining the parameters of the filter model, selecting a suitable excitation function such as a noise burst or impulse, and minimizing the coding distortion. During synthesis the filter coefficients and initial filter state are fetched and the excitation function is used to regenerate the wavetable. If the complexity of the filter and the excitation function can be minimized, the model approach can further reduce the storage needed for wavetable synthesis.

7. REFERENCES

[1] Mathews, Max (1969). *The Technology of Computer Music*, M.I.T. Press: Cambridge, MA.

[2] Dodge, Charles and Jerse, Thomas (1997). *Computer Music: Synthesis, Composition, and Performance*, 2nd ed., Shirmer Books: New York, NY.

[3] Bristow-Johnson, Robert (1996). “Wavetable Synthesis 101, A Fundamental Perspective,” Proc. 101st AES Conv., Preprint 4400.

[4] MIDI Manufacturers Association (2001). *GM Lite Specification and Guidelines for Mobile Applications*, MMA: La Habra, CA.

[5] Jayant, N.S., and Noll, Peter (1984). *Digital Coding of Waveforms*, Prentice Hall: Englewood Cliffs, NJ.

[6] Rossum, Dave (1989). “An Analysis of Pitch-Shifting Algorithms,” Proc. 87th AES Conv., Preprint 2843.

[7] MIDI Manufacturers Association (2001). *Complete MIDI 1.0 Detailed Specification*, version 96.1, MMA: La Habra, CA.

[8] Lindemann, Eric (1999). “Audio Data Decompression and Interpolation Apparatus and Method,” U.S. Patent Number 5,890,126.