INTERPOLATION ERROR IN WAVEFORM TABLE LOOKUP

Roger B. Dannenberg
School of Computer Science, Carnegie Mellon University
Pittsburgh, PA 15217 USA
rbd@cs.cmu.edu, http://www.cs.cmu.edu/~rbd

Abstract: Waveform tables are an important tool for synthesizing sound, but they introduce error which results in noise. Error is affected by the spectrum of the signal stored in the table. Error is reduced by increasing the table size and/or by increasing the quality of interpolation. Both of these also affect the signal computation cost. Table sizes required for a given signal-to-noise ratio are computed for different interpolation methods and spectral rolloffs. Execution times are then evaluated. Non-interpolated oscillators perform the best, but only if the storage and computation costs of the tables are not an issue. This and other tradeoffs are discussed.

1. Introduction

Software-based synthesizers are gaining in popularity because computers are becoming faster and cheaper at an exponential rate and because they offer tremendous flexibility. However, software is not the same as hardware. It is important to reconsider design choices rather than simply emulate existing hardware designs. A case in point it the table-lookup oscillator. General purpose processors are not particularly good at either random access to main memory or interpolation, but these operations are at the core of a table lookup oscillator. What is the best table size, and what is the best interpolation technique for a software implementation? This paper describes the factors that affect table-lookup oscillator performance, provides a methodology to study design tradeoffs, and presents some interesting results based on current processors.

Table lookup (including sampling) is an old but still important technique for music synthesis. Table lookup is used to generate sinusoids for additive synthesis, and fixed spectra for group additive, spectral interpolation, and vector synthesis. Samplers also use a form of table-lookup, and sample-rate conversion can be viewed as a generalization of table lookup. Thus, a fundamental operation for a variety of synthesis techniques is reading samples from tables.

The purpose of a table lookup oscillator is to output samples of a periodic function (call this *F* for now, and assume that the domain of *F* is the interval [0, 1)). A simple algorithm for computing samples, given frequency *hz*, sample rate *sr*, and function *F*, is:

```
phase = 0;
increment = hz / sr;
while (true) {
    output(F(phase));
    phase = phase + increment;
    // phase "wrap" for periodic output:
    if (phase >= 1) phase = phase - 1; }
```

Notice that *phase* and *increment* are floating point (or at least fixed-point numbers with fractional values), so *F* must be evaluated at arbitrary phases. This is where table lookup comes in. Equally spaced samples of *F* are stored in a table. To evaluate *F*(*phase*), we can simply choose the nearest sample, perform linear interpolation between the nearest samples, or perform some higher-order interpolation.[1] In any case, *table lookup yields an approximation of F*, and we are concerned with the error that is introduced.

Quantization noise is also important to consider, but for this study, we use floating point samples so that quantization noise is negligible.

2. Table Lookup Noise

Table lookup for sinusoids has been studied previously. (Moore 1977) However, spectrally rich signals complicate the analysis. There are at least two factors to consider. First, upper harmonics are effectively stored in smaller tables:

[1] In a practical implementation, the inner loop includes the table access and interpolation, and *phase* is computed such that its integer part is a direct table index (Roads 1996).

the Nth harmonic has N complete periods within the table, so the table size is effectively scaled by 1/N. Smaller table sizes yield larger errors. Figure 1 illustrates the signal-to-noise ratio (SNR) of a table-lookup oscillator using linear-interpolation, with 1 through 64 equal-amplitude harmonics in a table with 1024 entries. Notice that the SNR falls as the number of harmonics increases. This is largely because the table size for the 64th harmonic is effectively only 1024/64 = 16 samples.

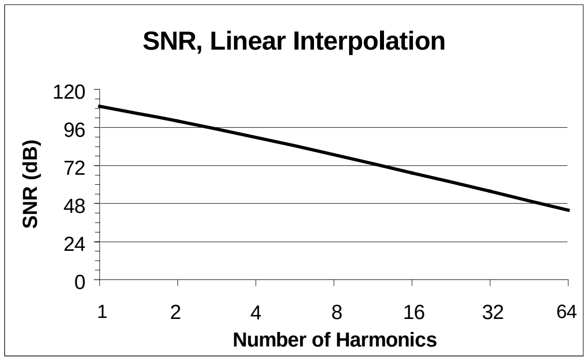

Figure 1. SNR falls as the number of harmonics increases.

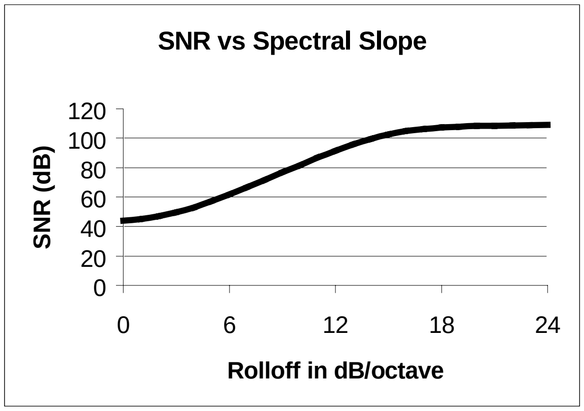

Figure 2: SNR increases as the spectral rolloff increases.

The second factor is that the amplitudes of upper harmonics typically fall rapidly with harmonic number. Smaller amplitudes give rise to smaller errors, potentially compensating for the smaller effective table size. In Figure 2, the SNR is computed for a 1024-sample table containing 64 harmonics, where the spectral slope is varied from 0 dB/octave to 24 dB/octave. Notice that with a large negative spectral slope (called the *rolloff*), the upper harmonics are lower in amplitude and the resulting waveform is "smoother," so the interpolation noise is less.

2.1 Estimating SNR

Throughout this work, the reported SNR is estimated as follows. The function *F* is computed as a weighted sum of sinusoids (computed using floating point arithmetic). *F* is evaluated at each point corresponding to a table entry. Then, table lookups are performed at equal intervals. For example, if the table size is 100, then *N* = 1000 lookups are performed corresponding to table offsets of 0.0, 0.1, 0.2, … 99.8, and 99.9. At each table lookup, the "true" value of *F* (limited by floating point resolution) is computed. *F* is the signal, and the difference between *F* and the table lookup value is the error (noise). The SNR in dB is estimated by:

$SNR = 20\log_{10} \sqrt{\sum F_i^2 / N} \Big/ \sqrt{\sum \Delta_i^2 / N}$, where Δ is the difference between *F* and the table lookup value.

3. Table Size

Perhaps the most interesting question is, given a spectral slope and some number of parameters, how large should the table be to achieve a certain SNR? Software synthesizers can have arbitrary table sizes, so it is possible to choose an appropriate size based on the spectrum. Figure 3 presents the required table size to achieve SNR's of 60, 72, 84, and 96 dB for 32 harmonics, with rolloff varying from 0 to 24 dB/octave. Table sizes range from 62 to 10402 samples! Note that at 62 samples, the 32nd harmonic actually aliases, but the amplitude is so low due to rolloff, this aliasing does not contribute significant error.

Linear interpolation is not the only option. In theory, any sampled signal can be recovered with arbitrarily low error provided that the signal's sample rate is higher than twice its highest frequency component. In the case of tables, the table should have at least twice as many samples as harmonics. In practice, "proper" interpolation requires many expensive memory operations and as many arithmetic operations, but even a short, low-quality interpolator should do better than linear interpolation. A less expensive (but less accurate) technique is polynomial curve fitting. A quadratic equation fit to three samples is more accurate than a linear equation fit to two samples. An equation for quadratic interpolation is:

$$F((i+p)/ts) = f_i + p\cdot((2-p)\, f_{i+1} - ((3-p)\, f_i - (1-p)\, f_{i+2})/2),$$

where *ts* is the table size, *i* is the integer part and *p* is the fractional part of the table offset, and f_i is the i^{th} table entry. For convenience, two “extra” table entries are added: $f_{ts} = f_0$ and $f_{ts+1} = f_1$. Note that the argument to *F* varies from 0 to 1 as described in the introduction.

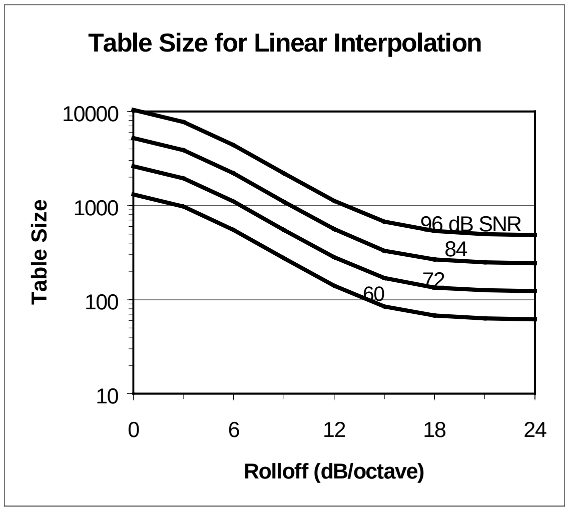

Figure 3. How large should a table be for linear interpolation?

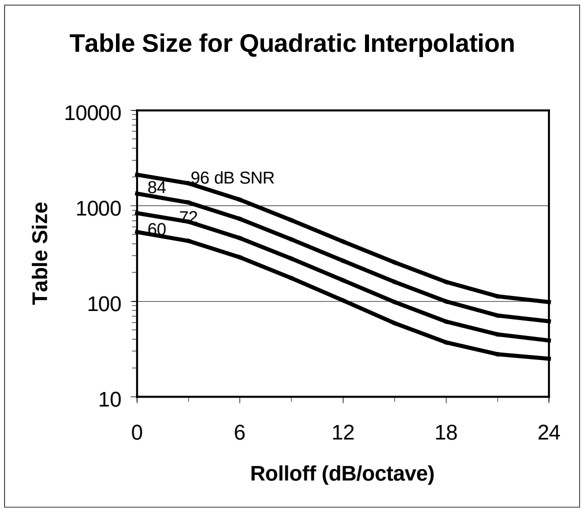

Figure 4. How large should a table be for quadratic interpolation?

Figure 4 is similar to Figure 3, except that quadratic interpolation is used. The reduction in table size ranges from 28% (12 dB/octave rolloff, 60 dB SNR) to 80% (0 or 24 dB/octave, 96 dB SNR).

No interpolation is another possibility. In other words, the phase is simply rounded to the nearest sample in the table. Since truncation is the standard floating-point-to-integer conversion, it is convenient to simply add 0.5 to the initial phase to accomplish rounding. Figure 5 is similar to Figures 3 and 4, except that no interpolation is used. Notice that the required table sizes are much larger than for linear interpolation (factors range from 13 to 239). The largest table size is 2,170,489 for 32 equal-amplitude harmonics and 96 dB SNR).

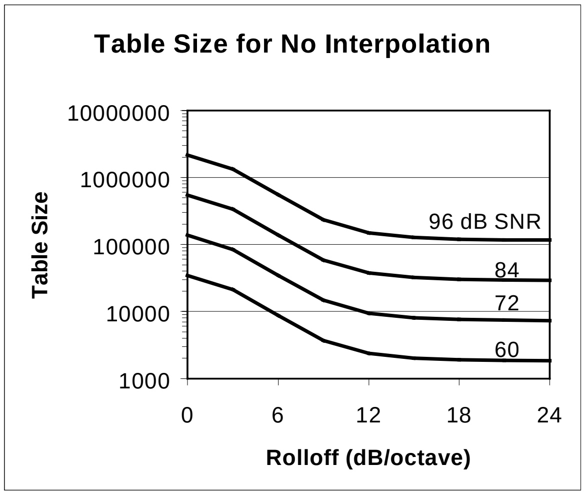

Figure 5. How large should a table be for no interpolation?

4. Execution Speed.

Using higher-order interpolation requires more processing power. Table 1 shows the amount of processing time required per sample to compute one sample using different interpolation techniques. These measurements were made using a single oscillator with a table size of 512. Notice that the cost of interpolation is not proportional to the number of arithmetic operations. This is because arithmetic operations are only a part of the computation, and parallelism in the CPU allows some of these operations to overlap.

Since higher-order interpolation conserves table space, it allows more tables to fit in the cache. The resulting savings in memory access time could potentially offset the extra arithmetic operations. A fair comparison, then, would use large tables for no interpolation and small tables for quadratic interpolation.

Interp. Method	Arith. Ops	Loads and Store	P133		PPro200	
			Time	Ratio	Time	Ratio
None	3	2	410 ns	1.00	324 ns	1.00
Linear	7	3	550 ns	1.35	400 ns	1.23
Quadratic	15	4	730 ns	1.80	498 ns	1.54

Table 1. Execution time for different interpolation methods. Table size = 1024.

Interp. Method	Arith. Ops	Loads and Store	Table Size	P133, 16 tables		P133, 125 tables	
				Time	Ratio	Time	Ratio
None	3	2	9428	500	1.00	460	1.00
Linear	7	3	282	570	1.14	550	1.20
Quadratic	15	4	166	770	1.55	730	1.59

Table 2. Execution time for different interpolation methods. Table size as shown.

Table 2 shows computation time per sample for 16 tables. The table sizes were arbitrarily chosen to give 72 dB SNR with a 12 dB/octave rolloff using the various interpolation techniques. These results should be *highly* dependent upon cache size, machine architecture, and table size, which in turn depends upon the spectral rolloff and the desired SNR. In this particular experiment, performance decreased when multiple tables were used, but using even more tables did *not* further increase the per-sample execution time.

5. Discussion.

In all cases the non-interpolated oscillator is fastest. Notice, however, that when table sizes are chosen to give results with equal SNR, the relative advantage of non-interpolated oscillators is reduced. For the particular parameters chosen for this experiment, the non-interpolated oscillator is only about 1.55 times as fast as the quadratic interpolation oscillator, and only 1.14 times as fast as the linear interpolation oscillator.

These times do not include the times to build the tables, which are much larger for the non-interpolated oscillator. If table construction time is taken into account, interpolating oscillators may in fact have an advantage. If tables are constructed by summing sinusoids, then it costs much more to generate a table sample than to access the table. For example, if it costs 20 times as much to generate a table sample as to access it with the non-interpolating oscillator, if table sizes are 8192 for non-interpolating tables and 128 for quadratic interpolating tables, and if quadratic interpolation is 1.6 times the cost of non-interpolating access, then a table would have to be used for $(8192 - 128) \times 20 / 1.6 = 100{,}800$ samples, or 2.3s of audio at a 44100 *hz* sampling rate. Otherwise, quadratic interpolation is faster (and linear interpolation would be faster still).

6. Future Work

This work can be extended to study sample rate conversion, including pitch shifting and sample-based synthesis. What degree of oversampling would be required to achieve acceptable SNR's? This work ignores the effects of human perception, including masking. Is SNR the right measure, and if so, what SNR is really necessary? Can this be confirmed with listening tests?

7. Conclusion

We have demonstrated that the table size required to achieve a given SNR is highly dependent upon the spectrum and the interpolation method. Software-based synthesizers should choose a table size and interpolation method that optimizes performance. Since musical spectra often exhibit significant rolloff, the performance of simple interpolation techniques can be quite good for table-lookup and other sample-rate conversion applications.

References

Moore, F. R. 1977. "Table lookup noise for sinusoidal digital oscillators." *Computer Music Journal* 1(2), pp26-29. Reprinted in C. Roads and J. Strawn, eds. 1985. *Foundations of Computer Music.* MIT Press. pp. 326-334.

Roads, C. 1996. *The Computer Music Tutorial.* Cambridge: MIT Press, p. 93.