Wavetable Acceleration Using Intermediate Buffering and Program Code Trimming

Steven D. Trautmann
Texas Instruments Tsukuba Research and Development, Japan
steve@trdc.ti.com

Abstract

The main computational expenses in wavetable synthesis have to do with the interpolation used for pitch shifting and the filtering applied to vary the timbre. Special intermediate buffering techniques can be used to reduce the computation cycle per PCM sample. The basic technique is to occasionally rewrite the wavetable after applying some filtering and interpolation to reduce the overall expense of these operations. If the new wavetable is itself refiltered to produce a continual darkening of the sound, the operation has some similarities to the Karplus-Strong plucked string algorithm, but instead of continuously filtering the wavetable, the entire wavetable is refiltered only occasionally. In order to reduce computation to a minimum, many sound engines can be used for wavetable which are optimal for particular situations. These sound engines are very similar except for a few operations. In order to conserve program memory, it is helpful to have just one basic sound engine coded, in which some machine code is rewritten on the fly before execution.

1 Introduction

Wavetable synthesis here refers to the long used technique of storing values of a single or few periods of the fundamental frequency of a sound in a table which is looped with some interpolation, scaling and filtering to create a sustained tone with some change in pitch, amplitude and timbre over time. Combined with sampling and longer looping sections it forms the basis of what is commonly referred to as simply "wavetable". In order to conserve memory in a constrained systems (many systems used 1 to 3 Megabytes for all 128 General MIDI instruments plus extra instruments) each sound sample typically covers a range of frequencies. In addition, samples are combined (layered) to form additional instruments without using additional memory. Also samples tend to be short, and often consist of a shortened attack which moves quickly into a loop at the fundamental frequency. In order to get the most out of such short, sparse samples, it is desirable to have robust methods of pitch shifting and timbre adjustment through filtering. However another constraint is processing time, which must be kept to a minimum, and this does not allow very extensive interpolation and filtering to be used.

2 Rewriting the Wavetable

Since most samples are short due to memory constraints, a great deal of time is spent in the looping section of the sample. Depending on instrumentation this can be well over 90% of the time. Also most of the looping sections are just one period of the fundamental frequency. Thus acceleration of looping sections (which are wavetables) accelerates the whole system. The most expensive computations involved in producing one PCM sample of output from a sound sample are interpolation (used to change the pitch) and filtering (to vary the timbre and prevent aliasing).

2.1 Rewriting at a new pitch

In cases where there is no vibrato, pitch shifting is not needed if the looping section is at the correct pitch already. Eliminating pitch shifting entirely obviously saves the most computation time, so for instruments like piano and harpsichord, which do not normally have vibrato, it makes sense to shift the pitch of the wavetable once and not need to do any subsequent pitch shifting. Unfortunately, to really benefit from this it is necessary to have the wavetable be an integer number of samples, which means arbitrary pitches are not possible at a set sampling rate. Fortunately it was found that by limiting the total wavetable size to a maximum of 1024 samples and allowing multiple periods of the fundamental frequency to be represented within this space, it is possible to create looping sections which approximate the desired pitch to within the JND for fundamental frequencies above about 40 Hz, assuming an output sampling rate of 44.1 kHz. (For example, the best approximation to the desired output frequency might be to write 7 periods consuming 942 samples and loop this
to achieve an output with no subsequent pitch shifting and within the JND of the desired output pitch).

2.2 Rewriting for drop sample

In the case where there is vibrato expected, such as violin or saxophone, the pitch change is too frequent to allow rewriting the wavetable at the desired pitch to be effective. Next to no interpolation, the method of 'drop sample' where a PCM sample is occasionally dropped or repeated is the fastest. However, when drop sample is used on a very short loop, a great deal of noise can be created due to its relative inaccuracy. To be able to use the drop sample method, it makes sense to expand the wavetable section to produce more accuracy whenever a desired location's value is taken as the nearest sample. Essentially this oversampling allows more accurate values to be chosen since the waveform is sampled in more places. This wavetable expansion to enable use of drop sample interpolation can either be done on the fly using a much more accurate multipoint interpolation method, or a set of expanded wavetables can be stored as part of the original sample set. The latter option does require some additional memory, but saves some additional computation. Either way the result is fast interpolation without undue noise.

2.3 Rewriting with filtering

Filtering is another computationally expensive function which, when used, is required for every output PCM sample. Usually the filter itself is only slowing varying over time. Thus, multiple periods of the fundamental frequency may receive essentially the same filtering each time before any noticeable change accrues. In this case it makes sense to filter the wavetable itself occasionally, and loop this prefiltered wavetable while no noticeable change is taking place. When this is done, no subsequent filtering is required, saving much computation. Also, higher quality filtering can be used since the filtering is only done rarely. When filtering is done every sample, time constraints often allow only a one or two-pole filter to be used, reducing the number of filtering options. When the filter does not change much over time, it is possible to invest computation at the beginning so that a more complex filter can create a sharper cutoff, or better Q, and still the overall computation will be less than employing a one-pole filter to every sample. Finally, subsequent filtering of the wavetable may be done from the previous pre-filtered wavetable rather than from the original, in order to further reduce computation when, for instance, the sound is just becoming darker. In this case the scenario has some resemblance to the Karplus-Strong plucked string algorithm, which also can be looked at as a wavetable which is being re-written after being filtered each time it is looped.[1][2] However, in this case the filtering to the wavetable is only applied occasionally, rather than every period. This can result in a more gradual and easily controlled darkening of the wavetable, but strains the relationship to a physical model of a string.

3 Rewriting Program Code

In addition to rewriting the actual wavetable for accelerating computation, it is also helpful to have different types of sound engines to perform the looping of the wavetable with various minor differences to accelerate particular situations. Often these different sound engines have only a few lines of differing assembly code. Since many slightly varying sound engines can be used to take advantage of constant amplitude, center panning position, etc., it can make sense to edit the sound engine code on the fly before execution in systems with constrained program memory space.

4 Conclusion

In a typical commercial wavetable system, memory and computation are severe constraints. Much time is spent looping a wavetable for many instruments. Certain redundancies related to interpolation and filtering occur in these looping sections each time through. Rewriting the wavetable looping section can eliminate many of these redundancies reducing overall computation time.

5 Acknowledgments

The author would like to thank Texas Instruments and in particular Raj Pawate for supporting this work and Ngai-Man Cheung for his assistance.

References