Real-Time Generation of Sound

from Parameters of Additive Synthesis

Robert Strandh (strandh@LaBRI.U-Bordeaux.Fr)
Sylvain Marchand (sm@LaBRI.U-Bordeaux.Fr)

SCRIME
LaBRI, Université Bordeaux I
351, cours de la Libération
F-33405 Talence cedex – France

Abstract

We describe a system for generating sounds in real-time. The input to the system is a flow of parameter values. These values control the frequency and amplitude of a bank of oscillators. The time between two sets of parameter values in the flow is much larger than the time between two samples of the resulting sound. The sound generation system must compute the instantaneous value of each oscillator and then sum the results.

The main problem is then to get sufficient performance out of the computation of each instantaneous oscillator value. Sine tables were ruled out because they either require interpolation (which is slow) or massive amounts of memory (essentially one table for each frequency).

A well-known formula exists for incrementally computing the samples of an oscillator, given any frequency and any amplitude, with only one floating-point multiplication and one floating-point addition per sample. This formula has two major problems. The first is that the parameters are fixed, whereas they need to vary, and the second is that numeric imprecisions may cause the values to drift. Other formulae exist that handle both these problems, but they are more expensive to compute.

The solution presented in this paper is able to adapt the fast formula so that the parameters may evolve and that drift is avoided. With this method, we are able to generate roughly 2 oscillators per MHz of clock speed on a Pentium II processor. In other words, we obtain the simultaneous generation of around 800 oscillators on a 400 MHz machine.

1. Introduction

In the InSpect system [1], we are able to extract very precise information about the partials of a sound, for use with additive synthesis [2], using a high-precision Fourier analysis [3]. The output of such an analysis is a flow of parameter values for a bank of oscillators. Each oscillator in the bank is responsible for exactly one partial. As opposed to methods based on the Fast Fourier Transform (FFT), the frequencies of the oscillators are not fixed, but vary slowly according to the parameter flow. This representation for sound introduced by McAulay and Quatieri [4] is very expressive musically. It has already been successfully used in software packages like Lemur [5] and SMS [6]. Figure 1 illustrates this spectral representation.

To simplify the presentation in this paper, we may assume that the instantaneous values of the parameters are found by linear interpolation between the values present in the flow. Other systems are possible. In particular, we are investigating whether it would be useful to consider the parameter values as the samples of a contiguous signal with a frequency of at most half the one defined by the time between parameter values.

The problem, then, is to find a very fast method for generating the sequence of samples for each oscillator with as few operations as possible. There are essentially two possibilities, although other techniques [7] exist when the oscillator parameters vary extremely slowly. The first one is based on table lookup and the second based on incremental computation of a sample based on the previous few samples.

In the past, table lookup was a reasonably fast method. Memory was relatively fast and arithmetic, especially on floating-point values, was much slower. As processors rapidly became faster and main memory remained roughly the same speed, this technique became less interesting. At the same time, progress was made with respect to the speed of floating-point arithmetic. On a modern processor, a floating-point addition can be done in 1 cycle (pipelined) with a latency of around 3 cycles, and a floating-point multiplication can be done in 1 or 2 cycles with a latency of around 5 cycles (for the Pentium family processors). The trend is toward even faster arithmetic and higher clock speeds whereas memory speed still remains roughly the same.

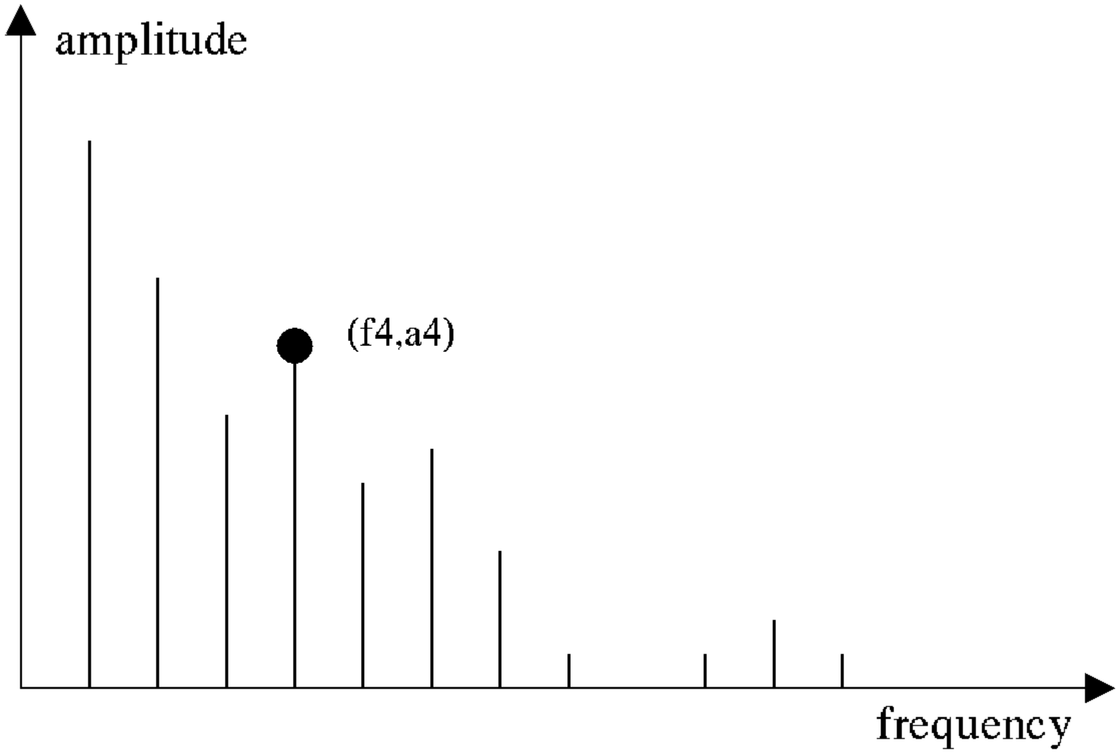

Figure 1: Spectral representation, at time t, of a quasi-periodic sound consisting of 11 partials.

A natural choice, then, seems to be a method based on incremental floating-point arithmetic. There is a well-known formula of trigonometry [8, 9] that allows us to compute the sequence a.sin(ω.i) for each i incrementally, where a is the amplitude and ω is proportional to the frequency f. Using this formula, the value of some sample s[i] can be expressed as a function of the two previous samples s[i-1] and s[i-2] like this:

$$s[i] = 2.\cos(2\pi f\, T).s[i-1] - s[i-2] \quad \text{(T is the sampling period in seconds)}$$

As we can see, this formula only requires one multiplication and one addition. However, this formula is not very well adapted to our needs. It has two major problems. The first is the numeric instability. The second is that our parameters vary over time. In theory, both the amplitude and the frequency can vary slightly between each new sample computed. The reason is that parameter evolution is expressed as an interpolation between values in the parameter flow. So even though we compute thousands of samples between two parameter values in the flow, because we interpolate, we must be prepared to adjust both the frequency and the amplitude of our oscillators at each sample.

The necessity for such adjustments seems to make this formula useless. There are similar formulae that take into account the possibility of linear evolutions of the amplitude, but we are unaware of any fast formula that takes into account linear variation of the frequency as well. Furthermore, if we change from linear interpolation, even those formulae would become useless. In any case, formulae that take into account parameter variation are much more expensive to compute, at least double the cost of the one above.

In the next section, we present our solution, which represents a very small percentage overhead compared to the fastest formula shown above.

2. Our method

Our method is based on two crucial observations. First, our experiments indicate that abruptly changing the amplitude by a reasonable amount does not introduce any audible noise or distortion. Similarly, changing the frequency by a reasonable amount also does not introduce any audible impact on the quality of the sound. Notice that an abrupt change in phase might have such an impact. But as long as the phase stays the same, we may change the frequency by quite a lot. The second observation is that the slope of our parameter values, i.e. their speed of variation, is going to be relatively small compared to what would still go undetected.

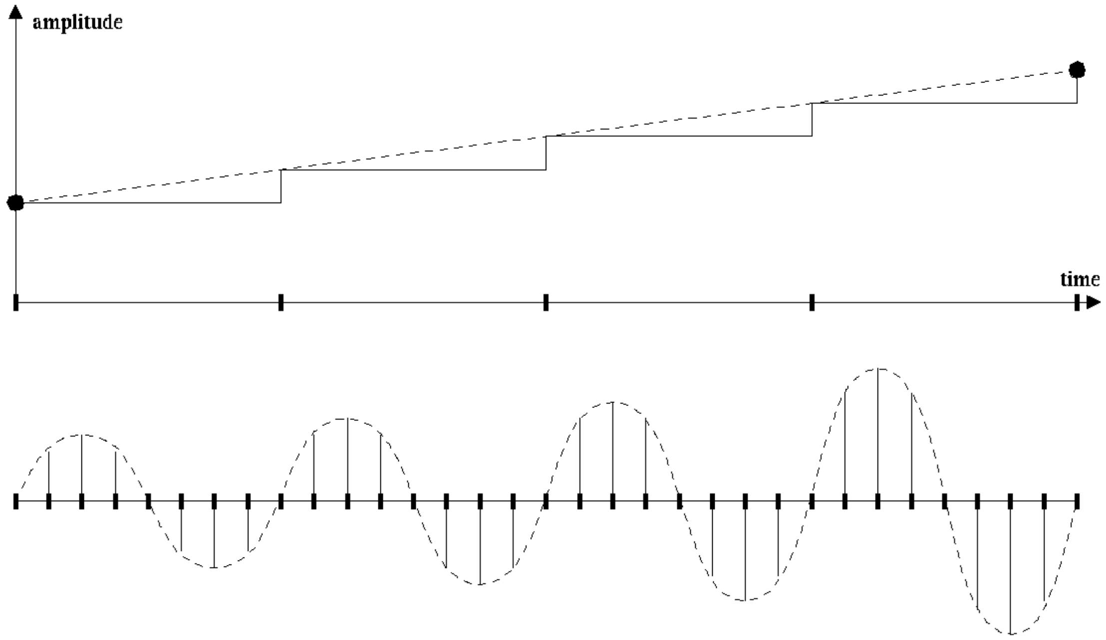

Figure 2: Variation of the amplitude of an oscillator and the resulting audio signal.
Between two parameter changes, some interpolated values are computed (4 in this example).
And between two interpolated values, many samples are computed…

With this in mind, we can now present our method. Based on two consecutive parameter values, determine the slope of the evolution of the parameter values. The slope is defined to be the difference in parameter values divided by the number of signal samples between two parameter points. In our case, there can be up to 1000 samples in such an interval for a sampling rate of 44100 Hz. From experimental data, we know the maximum allowable difference in parameter values we can have without creating any audible distortion or noise. Divide this value with the slope previously obtained. The result of the calculation is the number of samples that can be generated without any adjustment of the parameters.

Experiments show that we frequently get intervals of 100 samples or more between necessary adjustments of parameter values. Thus, even if the computation to adjust parameter values is considerably more expensive than that of generating a sample, we are still within a few percents of extra cost compared to maximum speed when parameters do not evolve at all. Figure 2 shows how our method handles the variations of the parameters.

3. Numeric imprecisions

It is well known that the fast formula that we use for generating samples does not behave very well with respect to loss of precision in numeric calculations. If iterations are carried out for a long time, drift in the value of the frequency can cause the sound to be severely distorted.

Fortunately, such distortion will only be introduced after a considerable number of iterations, in particular since we carry out our calculations in double precision.

Since our method require us to adjust our parameters every so often, say every 100 samples or so, we take advantage of this periodicity to recompute our initial samples from scratch, i.e. using trigonometric functions. This way, we not only adjust our parameters, but also compensate for any possible drift due to numeric imprecision.

Should computing the trigonometric functions be expensive at this frequency, it can be done at a much lower frequency than is required to adjust parameters. It just happens to be convenient to do both at the same time.

4. Possible improvements

We already mentioned that it is possible to adjust (for instance) the amplitude by a reasonable amount without introducing any distortion. But we can actually do even better.

Such adjustments are more audible the greater the absolute value of the current sample to be computed. The reason for this is that the difference in amplitude is multiplied by the value of the signal. The smaller the signal, the smaller the (potentially audible) difference in the amplitude difference.

The frequency parameter has the inverse problem. An abrupt change in the frequency is more likely to be audible when the value of the signal sample is small. The reason for this is that the derivative of the signal changes as a result of a change in frequency, and the derivative is zero when the absolute value of the signal is the greatest. These phenomena are shown in figure 3.

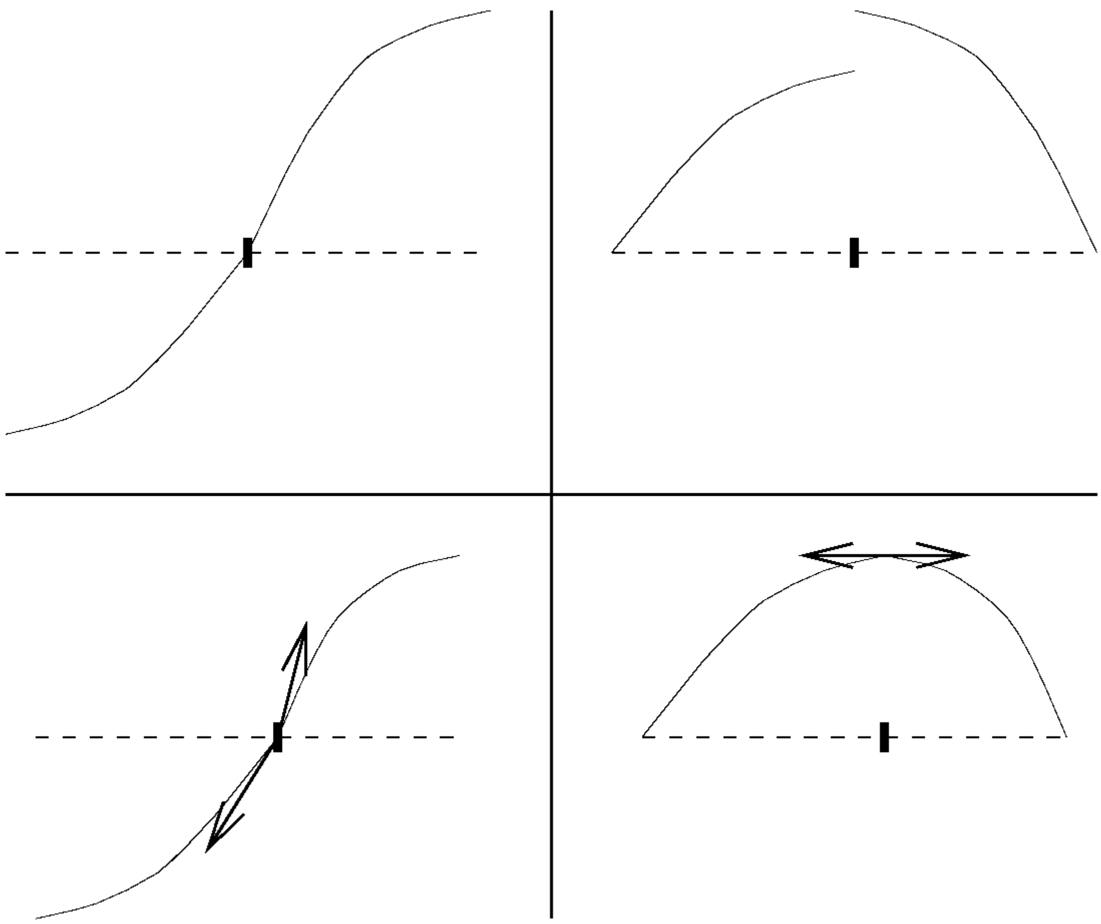

Figure 3: Changing the amplitude (top) and the frequency (bottom), either when the signal is minimal (left) or maximal (right). It appears that the left case is better for the amplitude while the right one is better for the frequency.

With this in mind, we can improve the situation even further. The idea is then to delay changes in amplitude until the signal value is close to zero, and to delay changes in frequency until the absolute value is maximal. While we have not tested this method yet, we believe that it will allow us to obtain performance very close to the maximum theoretical value for fixed parameter values.

Naturally, if we had to test the value in each iteration to determine whether it is close to zero or, on the contrary, close to maximal, we would lose big. Such a test would take time comparable to the computation of a sample, which would slow us down by a factor close to 2.

Fortunately, we can avoid that problem, and here is how. When parameters are adjusted, we precompute the number of iterations before the next update is necessary. This computation makes sure that the phase after the next block of iterations is optimal. By optimal we mean that it should be close to 0 or π for the amplitude and close to $\pi/2$ or $3\pi/2$ for the frequency. The computation of the samples is then started for a known number of iterations. When that computation finishes, we know it is the optimal time to adjust the parameters.

To avoid a test for the loop counter in each iteration, the loop is unfolded a sufficient number of times that the time taken by this test is negligible.

5. Conclusions and future work

We have presented a method for fast additive synthesis. We have successfully obtained close to optimal performance, which for a Pentium II family processor is around 2 oscillators per MHz of clock frequency, which gives us 800 oscillators on a 400 MHz processor.

While we have not investigated this in detail, we believe that a number of oscillators not much larger than that is enough for nearly all possible sounds. The reason is that as the number of oscillators grows, the smallest distance between two oscillators is going to get smaller. When this distance is sufficiently small, either we get a psycho-acoustic phenomenon known as masking, or else, the two oscillators can be combined into one without altering the sound.

It would then seem that there is an upper bound on the number of such oscillators that we need in order to produce most sounds. We are already very close (if not already past) this number with the method presented in this paper.

Further research includes exact determination of phenomena such as masking and other related ones that may help us decrease the number of oscillators needed, and thereby increasing performance.

6. References

[1] Sylvain Marchand. *InSpect Software Package*. URL http://www.scrime.u-bordeaux.fr, 1998.

[2] James A. Moorer. Signal Processing Aspects of Computer Music – A Survey. *Computer Music Journal*, 1(1):4–37, 1977.

[3] Sylvain Marchand. Improving Spectral Analysis Precision with an Enhanced Phase Vocoder using Signal Derivatives. *Proceedings of the Digital Audio Effects Workshop (DAFX'98, Barcelona)*, pages 114–118, 1998.

[4] Robert J. McAulay and Thomas F. Quatieri. Speech Analysis/Synthesis Based on a Sinusoidal Representation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34(4):744–754, 1986.

[5] Kelly Fitz and Lippold Haken. Sinusoidal Modeling and Manipulation Using Lemur. *Computer Music Journal*, 20(4):44–59, 1996.

[6] Xavier Serra. *Musical Signal Processing*, chapter « Musical Sound Modeling with Sinusoids plus Noise », pages 91–122. Studies on New Music Research. Swets & Zeitlinger, Lisse, the Netherlands, 1997.

[7] Adrian Freed, Xavier Rodet and Philippe Depalle. Synthesis and Control of Hundreds of Sinusoidal Partials on a Desktop Computer without Custom Hardware. *Proceedings of the International Computer Music Conference (ICMC'93, Tokyo)*, 1993.

[8] J. W. Gordon and J. O. Smith. A Sine Generation Algorithm for VLSI Applications. *Proceedings of the International Computer Music Conference (ICMC'85)*, pages 165–168, 1985.

[9] Julius O. Smith and Perry R. Cook. The Second-Order Digital Waveguide Oscillator. Technical Report. CCRMA, Music Department, Stanford University, USA.