Sound Synthesis Theory/Oscillators and Wavetables

Oscillators and Wavetables

Oscillators

An *oscillator* is a repeating waveform with a fundamental frequency and peak amplitude and it forms the basis of most popular synthesis techniques today. Aside from the frequency or pitch of the oscillator and its amplitude, one of the most important features is the shape of its waveform. The time-domain waveforms in **Fig. 5.1** show the four most commonly used oscillator waveforms. Although it is possible to use all kinds of unique shapes, these four each serve a range of functions that are suited to a range of different synthesis techniques; ranging from the smooth, plain sound of a sine wave, to the harmonically rich buzz of a sawtooth wave.

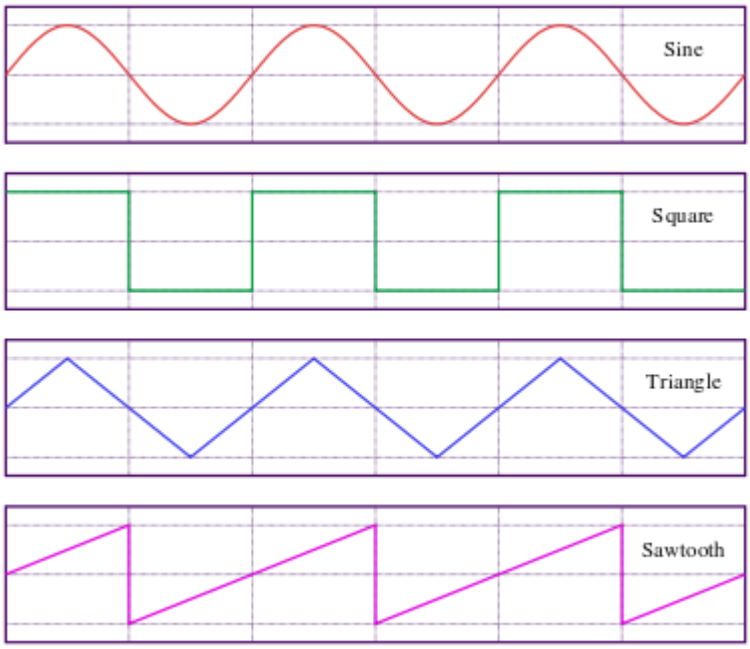

Figure 5.1 Sine, square, triangle, and sawtooth waveforms

Oscillators are generally controlled by a keyboard synthesiser or MIDI protocol device. A key press will result in a MIDI note value which will be converted to a frequency value (Hz) that the oscillator will accept as its input, and the waveform period will repeat accordingly to the specified frequency. From here, the sound can be processed or manipulated in a variety of ways in the synthesizer or program to enrich or modify the sound further.

Generating oscillator waveforms

Sine wave

As mentioned previously, the sine wave can be considered the most fundamental building block of sound. The best way to generate an oscillator which produces this waveform is to make use of an inbuilt library or function in the system concerned. Many programming languages have standard mathematics libraries with many of the trigonometric functions represented. A cycle of a sine wave is 2π radians long and has a peak amplitude of $+/-1$, as shown in **Fig. 5.2**. In a digital system, the generated waves will be a series of equally-spaced values at the sample rate.

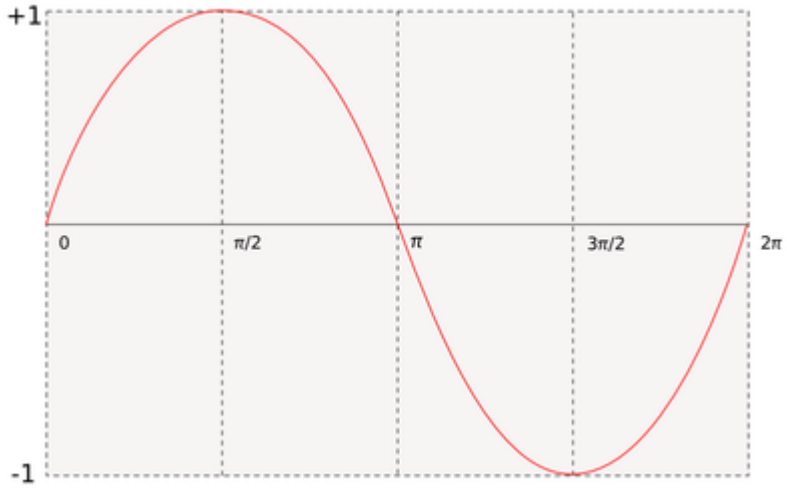

Figure 5.2 One cycle of a sine wave with phase 2π (radians).

With a sample rate of 44100 cycles per second, and a required cycle length of 1 second, it will take 44100 samples to get from 0 to 2π. In other words, we can determine the steps per cycle S from cycle length T:

$$S = T \cdot F_s$$

Where F_s is the sample rate. Each step will therefore take the following amount in radians:

$$\delta\phi = \frac{2\pi}{T \cdot F_s} \text{ or } \frac{2\pi f}{F_s}$$

Where f, in the second result, is the same result in terms of frequency. The importance of this is that it is possible to expand it into an algorithm that will be suitable for generating a sinusoidal wave of a user specified frequency and amplitude- effectively the simplest synthesizer possible! A sinusoidal wave can be generated by repeatedly incrementing a phase value by an amount required to reach a desired number of 2π length cycles a second, at the sample rate. This value can be passed to a sine function to create the output value, between the user specified peak amplitude.

```
Input: Peak amplitude (A), Frequency (f)
Output: Amplitude value (y)

y = A * sin(phase)

phase = phase + ((2 * pi * f) / samplerate)

if phase > (2 * pi) then
      phase = phase - (2 * pi)
```

The most important thing to note about this algorithm is that when the phase value has exceeded 2π it will subtract by one whole period. This is to ensure that the function "wraps" round to the correct position instead of going straight back to 0; if a phase increment *oversteps* 2π and resets to 0, undesirable discontinuities would occur, causing harmonic distortion in the oscillatory sound.

Square wave

The square wave cannot be generated from a mathematical function library so easily but once again the algorithm is particularly straightforward since it is constructed from straight line segments. Unlike the sine wave, square waves have many harmonics above their *fundamental frequency*, and have a much brighter, sharper timbre. After examining a number of different waveforms it will start to become apparent that waveforms with steep edges and/or abrupt changes and discontinuities are usually harmonically rich.

(Note that the following square, sawtooth, and triangle functions are "naive"; they are equivalent to sampling the ideal mathematical functions without first bandlimiting them. In other words, all of the harmonics above the Nyquist frequency will be aliased back into the audible range. This is most obvious when sweeping one of these waveforms into the high

frequencies. The aliased harmonics will move up and down the frequency spectrum, making "radio tuning" sounds in the background. A better method to produce waveforms for audio would be additive synthesis, or something like MinBLEPs. A properly-bandlimited waveform will have "jaggies" as you approach the discontinuities instead of piecewise straight lines.)

The square wave is constructed in a very similar fashion to the sine wave, and we use the same approach by cycling through a pattern with a phase variable, and resetting once we exceed 2π radians.

```
Input: Peak amplitude (A), Frequency (f)
Output: Amplitude value (y)

if phase < pi then
      y = A
else
      y = -A

phase = phase + ((2 * pi * f) / samplerate)

if phase > (2 * pi) then
      phase = phase - (2 * pi)
```

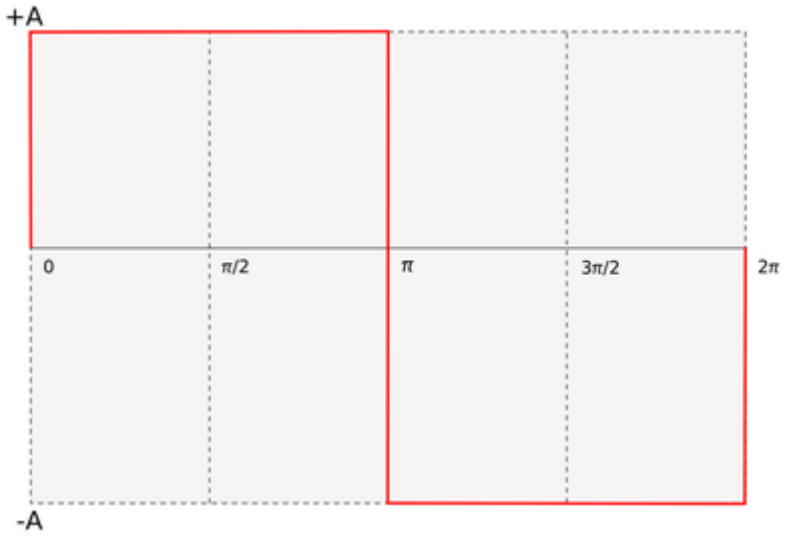

Figure 5.3 One cycle of a square wave with phase 2π (radians).

As is evident there is no reliance on an external function, the square wave can be defined by simple arithmetic, since it essentially switches between two values per cycle. One can expand on this by introducing a new variable which controls the point in the cycle the initial value switches to its signed value; this waveform is known as a *pulse wave*. Pulse waves are similar in character to square waves but the ability to modulate the switching point offers greater sonic potential.

Sawtooth wave

The sawtooth wave is more similar in sound to a square wave although it has harmonic decay and an appropriately "buzzy" timbre. It is constructed out of diagonal, sloping line segments and as such requires a line gradient equation in the algorithm. The mathematical form:

$$y = A - \frac{A}{\pi}\phi$$

Where A represents amplitude and ϕ is the phase. This can be incorporated into the algorithmic form as follows:

Figure 5.4 One cycle of a sawtooth wave with phase 2π (radians).

```
Input: Peak amplitude (A), Frequency (f)
Output: Amplitude value (y)

y = A - (A / pi * phase)
```

```
phase = phase + ((2 * pi * f) / samplerate)

if phase > (2 * pi) then
      phase = phase - (2 * pi)
```

Triangle wave

The triangle wave shares many geometric similarities with the sawtooth wave, except it has two sloping line segments. The algebra is slightly more complex and programmers may wish to consider consolidating the line generation into a new function for ease of reading. Triangle waves contain only odd-integer harmonics of the fundamental, and have a far softer timbre than square or saw waves, which is nearer to that of a sine wave. The mathematical form of the two lines segments are:

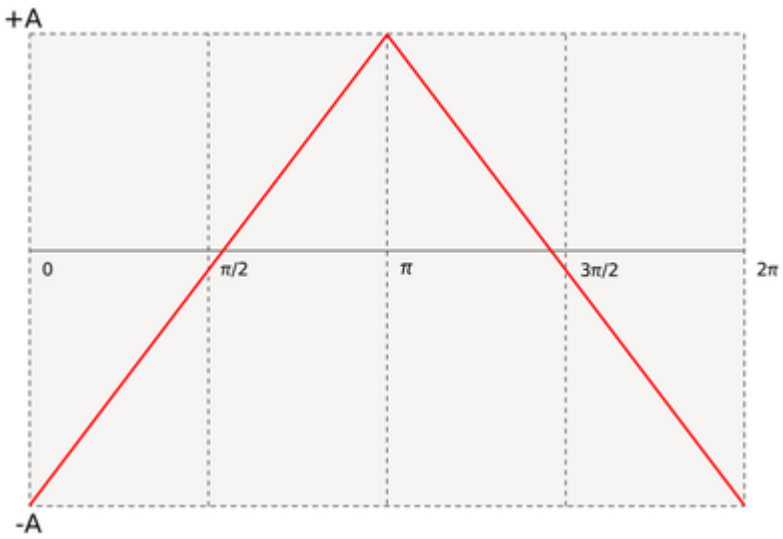

Figure 5.5 One cycle of a triangle wave with phase 2π (radians).

For 0 to π radians:

$$y = -A + \frac{2A}{\pi}\phi$$

For π to 2π radians:

$$y = 3A - \frac{2A}{\pi}\phi$$

The algorithm then, is similar to the previous examples but with the gradient equations incorporated into it. In the example algorithms presented here it is evident that a range of different waveshapes can be designed, but there is the realisation that the shapes can only be described as mathematical functions. Complex shapes may become very demanding due to the increased processing power for more complicated mathematical statements.

```
Input: Peak amplitude (A), Frequency (f)
Output: Amplitude value (y)

if phase < pi then
      y = -A + (2 * A / pi) * phase
else
      y = 3A - (2 * A / pi) * phase

phase = phase + ((2 * pi * f) / samplerate)

if phase > (2 * pi) then
      phase = phase - (2 * pi)
```

Wavetables

There may be a situation or a desire to escape the limitations or complexity of defining an oscillatory waveform using mathematical formulae or line segments. As mentioned before, this could be a concern for processing power, or simply the fact that it would be easier to specify the shape through an intuitive graphical interface. In cases like these, musicians and engineers may use *wavetables* to be their source oscillator. Wavetables are popular in digital synthesis applications because accessing a block of memory is computationally faster than calculating values using mathematical operations.

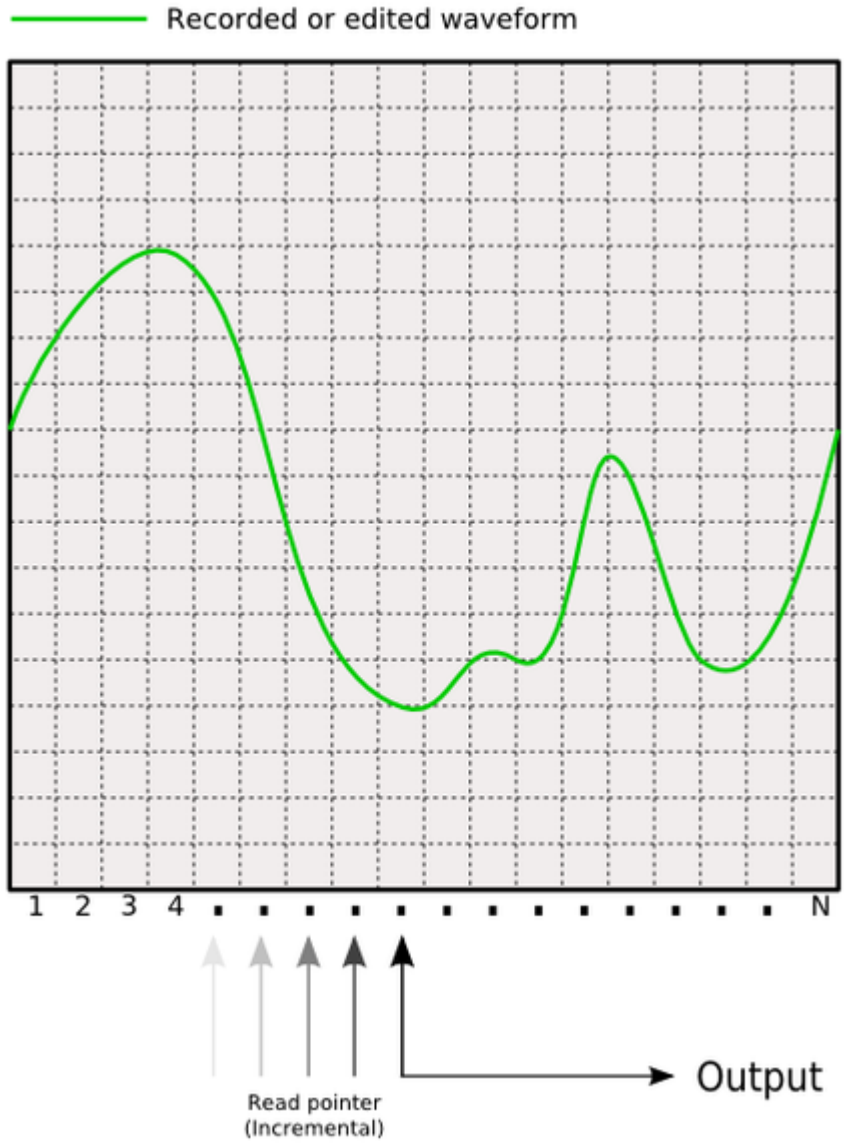

Figure 5.6 The basic structure of a wavetable oscillator.

The wavetable is in essence an array of *N* values, with values 1 through to N representing one whole cycle of the oscillator. Each value represents an amplitude at a certain point in the cycle. Wavetables are often displayed graphically with the option for the user to draw in the waveshape they require, and as such it represents a very powerful tool. There is also the possibility of loading a pre-recorded waveshape as well; but note that a wavetable oscillator is only a reference table for one cycle of a waveform; it is not the same as a *sampler*. The wavetable has associated with it a *read pointer* which cycles through the table at the required speed and outputs each amplitude value in sequence so as to recreate the waveform as a stream of digital values. When the pointer reaches the last value in the table array, it will reset to point one and begin a new cycle.

Using wavetables

The size of the wavetable and the sampling rate of the system determine what the fundamental frequency of the wavetable oscillator will be. If we have a wavetable with *1024* individual values and a sampling rate of *44.1* kHz, it will take:

$$\frac{1024}{44100} = 0.023$$

seconds to complete one cycle. As previously shown, frequency can be determined from $1/t$, giving us a fundamental frequency of:

$$\frac{1}{0.023} = 43.5 \text{ Hz.}$$

It therefore becomes apparent that, in order to change the frequency of our oscillator we must change either the size of the wavetable or the sampling rate of the system. There are some real problems with both approaches:

- Changing the *wavetable size* means switching to a different sized wavetable with the same, updated waveform. This would require dozens, hundreds, or even thousands of individual wavetables, one for each pitch, which is obviously totally inefficient and memory-consuming.
- Digital systems, especially mixers that combine several synthesized or recorded signals, are designed to work at a fixed *sampling rate* and to make sudden changes to it is once again inefficient and extremely hard to program.
- The sample rate required to play high frequencies with an acceptable level of precision becomes very high and puts high demand on the system.

One of the most practical and widely-used approaches to playing a wavetable oscillator at different frequencies is to change the size of the "steps" that the read pointer makes through the table. As with our previous example, our 1024-value wavetable had a fundamental frequency of 43.5 Hz when every point in the table was outputted. Now, if we stepped through the table every 5 values, we would have:

$$\frac{1024/5}{44100} = 0.0046 = 217.4 \text{ Hz.}$$

It follows from this a general formulae for calculating the required step size, *S* for a given frequency, *f*:

$$S = N\frac{f}{F_s}$$

Where *N* is the size of the wavetable and F_s is the sample rate. It is important to note that because the step size is being altered, the read pointer may not land exactly on the final table value *N*, and so it must "wrap around" in the same fashion as the functionally generated waveforms in the earlier section. This can be done by subtracting the size of the table from the current pointer value if it exceeds N; the algorithmic form of which can easily be gleaned from the examples above.

Frequency precision and interpolation

We must consider that some frequency values may generate a step size that has a fractional part; that is, it is not an integer but a rational number. In this case we find that the read pointer will be trying to step to locations in the wavetable array that do not exist, since each member of the array has an integer-valued index. There may be a value at position 50, but what about position 50.5? If we desire to play a frequency that uses a fractional step size, we must consider ways to accommodate it:

- **Truncation and rounding**. By removing the fraction after the decimal point we reduce the step size to an integer—this is *truncation*. For instance, 1.3 becomes 1, and 4.98 becomes 4. *Rounding* is similar, but chooses the closest integer—3.49 becomes 3, and 8.67 becomes 9. For simple rounding, if the value after the decimal point is less than 5, we round down (truncate), otherwise we round up to the next integer. Rounding may be supported in the processor at no cost, or can be done by adding 0.5 to the original value and then truncating to an integer. For wavetable synthesis, the only difference between truncation and rounding is a constant 0.5 sample phase shift in the output. Since that is not detectable—and neither an improvement nor a detriment—a decision between truncation and rounding comes down to whichever is more convenient or quicker.
- **Linear interpolation**. This is the method of drawing a straight line between the two integer values around the step location and using the values at both points to generate an amplitude value that interpolates between them. This is a more computationally demanding process but introduces greater precision.
- **Higher order interpolation**. With linear interpolation considered *first order* interpolation (and truncation and rounding considered *zero order* interpolation) there are many higher-order forms that are commonly used—cubic Hermite, Lagrangian, and others. Just as linear interpolation requires two points for the calculation, higher orders require even more wavetable points to be used in the calculation, but produce a more accurate, lower distortion result. *Sinc interpolation* can be made arbitrarily close to perfect, at the expense of computation time.

By increasing the wavetable size, the precision of the above processes becomes greater and will result in a closer fit to the idealised, intended curve. Naturally, large wavetable sizes result in greater memory requirements. Some wavetable synthesizer hardware designs prefer table sizes that are powers of two (128, 256, 512, 1024, 2048, etc.), due to shortcuts that exploit the way in which digital memory is constructed (binary).
