

The Tiny Sample Generator

Adrian Boeing, 2004

Introduction.

What?

Tiny Sample Generator is a minimalistic implementation of a synthesizer, which most closely resembles a modular analogue synthesizer. The synthesizer consists of various 'components' which can be joined via 'connections'. These concepts loosely represent different electronic 'modules' (such as amplifiers), and simple wires respectively.

Why?

Tiny Sample Generator is a small software synthesizer designed to use the least reasonable space in terms of code size, whilst still providing enough functionality to allow complex samples to be generated. The sample generator is intended to be used in conjunction with a tracker, which is why certain features are not present.

Tiny sample generator currently compiles into 6kb, and can be reduced to 3.5kb via compression. The sample format used by the synthesizer is itself very small, and designed to achieve a high level of compression.

Theory - Components.

The synthesizer consists of five different components:

- **Parameters:** These represent constant parameters, or variables used as controlling inputs into other components. In terms of analogue synthesis, this can be considered as a constant voltage being applied to a component.
- **Oscillators:** Oscillators produce an output signal that repeats at a specified frequency. Oscillators produce the fundamental signals that drive all the sounds within the synthesizer.
- **Offset Multipliers:** The offset-multiply component allows any signal to be multiplied and given an offset. This is equivalent to a voltage amplifier that also allows a DC offset to be applied to a signal.
- **Envelopes:** Envelopes modify the amplitude of a signal according to the length of time the signal has been active. They are somewhat similar to 'instruments' in a tracker
- **Filters:** A filter performs various 'modifying' operations on a signal that passes through it.

Oscillators:

There are five different selectable signal generators for the synthesizer. These determine the shape of the signal used to create a sound.

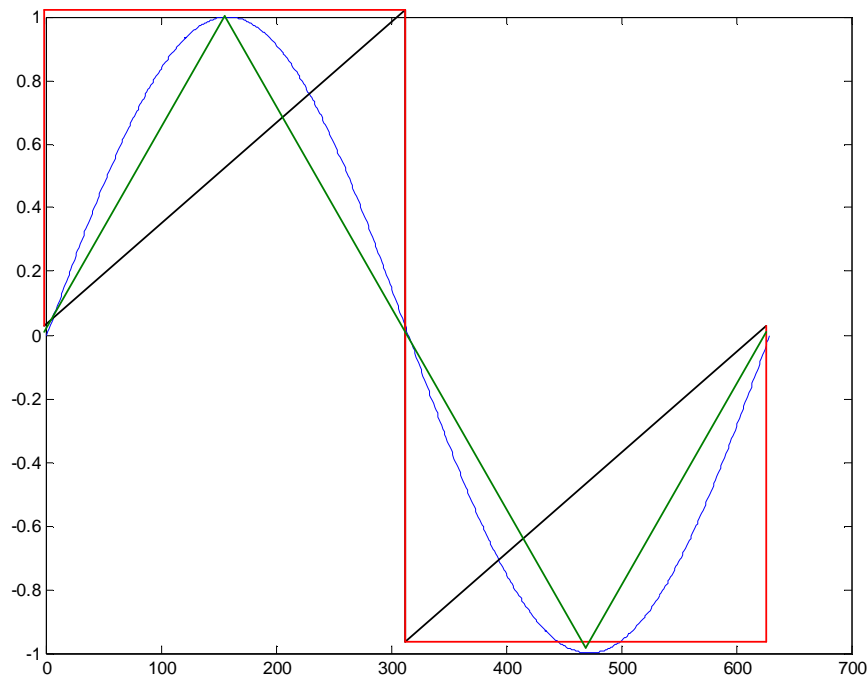


Figure 1. Signal Generator Functions

The ‘sine’ generator is a mathematical sine function, which is illustrated by the blue curve in Figure 1. Pure sine waves tend to have a very ‘soft’ sound.

The ‘square’ generator is illustrated in red. This tends to have a ‘hard’ sound.

The ‘sawtooth’ generator creates a “linear” wave, shown in black. The sounds produced by the sawtooth tend to be ‘sharp’.

The ‘triangle’ generator creates a “linear” wave similar to the sawtooth, except that the signal is more continuous. The ‘triangle’ wave is illustrated in green. It produces a harder sound than a ‘sine’ wave, but still softer than others.

An additional generator, “noise” is included, but not illustrated. This generates pseudo-random values for the signal representation. This sounds like “static” or, otherwise known as “white noise”.

Envelopes:

The synthesizer supports only one type of envelope, the ADSR envelope. The envelope contains 4 different sections: attack, decay, sustain, and release.

The attack determines how long the sound takes to get to full amplitude. Decay indicates how long the sound takes to get to the ‘sustain’ amplitude. The sustain determines the length of time the sound stays at the ‘sustain’ amplitude, and the decay determines how long the sound takes to get to zero amplitude (fade out).

The amplitude ‘envelope’ is illustrated in Figure 2.

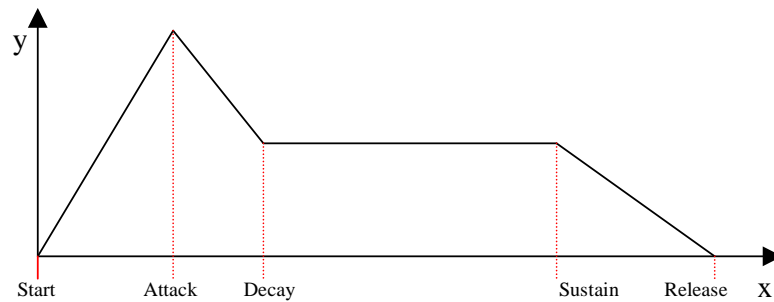


Figure 2. Amplitude Envelope

The 'x' axis represents time, and the 'y' axis represents the amplitude. The first red line indicates the starting time of the sample, the second line indicates the 'attack' time value, followed by the 'decay', 'sustain' and 'release' times respectively.

Offset Multiply:

The offset multiply component is easiest described mathematically:

$$\text{output} = \text{input} * \text{multiply_constant} + \text{offset_constant}$$

Equation 1. Offset Multiply

Filters:

There are a number of different filters that are part of the synthesizer:

- **Low Pass :** This filter is a second order resonant low pass butter worth filter. Given a specified frequency value (the cut-off value) only signals with a frequency that are roughly below the cut-off will pass through the filter. Thus, only 'low' pitch sounds will pass to the output. This is somewhat like using a graphic equalizer on a stereo to cut out all the high frequency sounds.
- **High Pass:** Similar to the low-pass filter, except it performs exactly the opposite operation. The high pass filter will remove all sounds in a signal lower than the specified cut-off frequency value.
- **Wave Shaper:** A wave shaper re-shapes the input sound wave to a different form. Wave shapers can be used to sharpen, or soften sounds. The wave shaper simply transforms one input value to another, different output value. Figure 3 illustrates the wave shaper with differing input values. The black line indicates a one-to-one mapping from input to output.

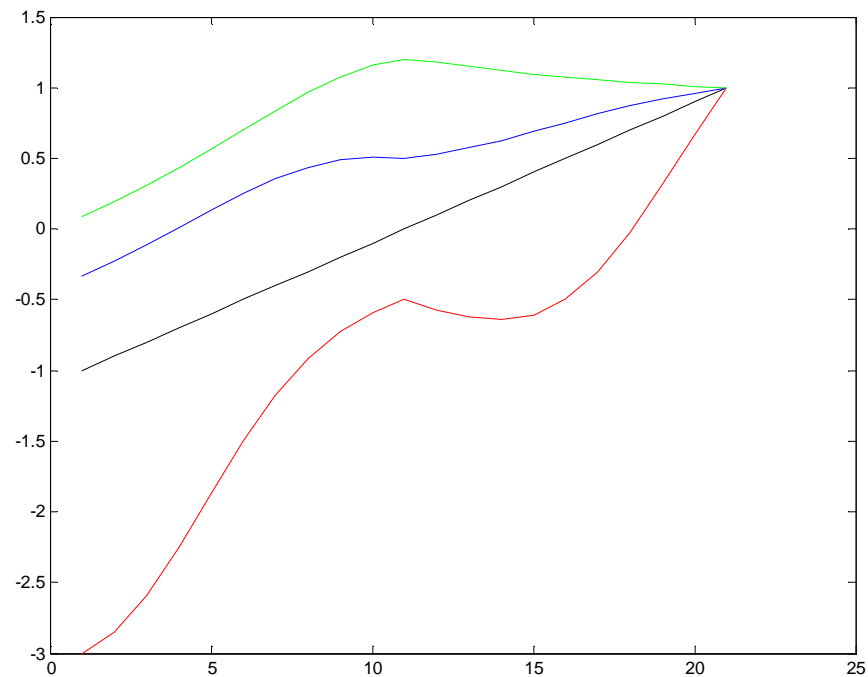


Figure 3. Waveshaper 1

In the above figure, the blue line represents Waveshaper1 with an input value of 0.5, The red line with an input value of -0.5, and the green line represents an overdrive value 1.2 (ie >1)

- **Smooth:** The smoothening effect smoothenes out a signal to reduce the effect of small clicking artifacts that may occur during the synthesis process. The smoothing effect simply takes a moving average of previous output values to achieve this effect.
- **Modulation:** Modulation is the effect of ‘multiplying’ two signals together. (Note: ring modulation filter is simply modulating a sound with a sine wave). Figure 4 illustrates modulation. The two input signals (The offset sawtooth wave (blue), and the sine wave (red)) are multiplied together to give the resulting wave shown in green.

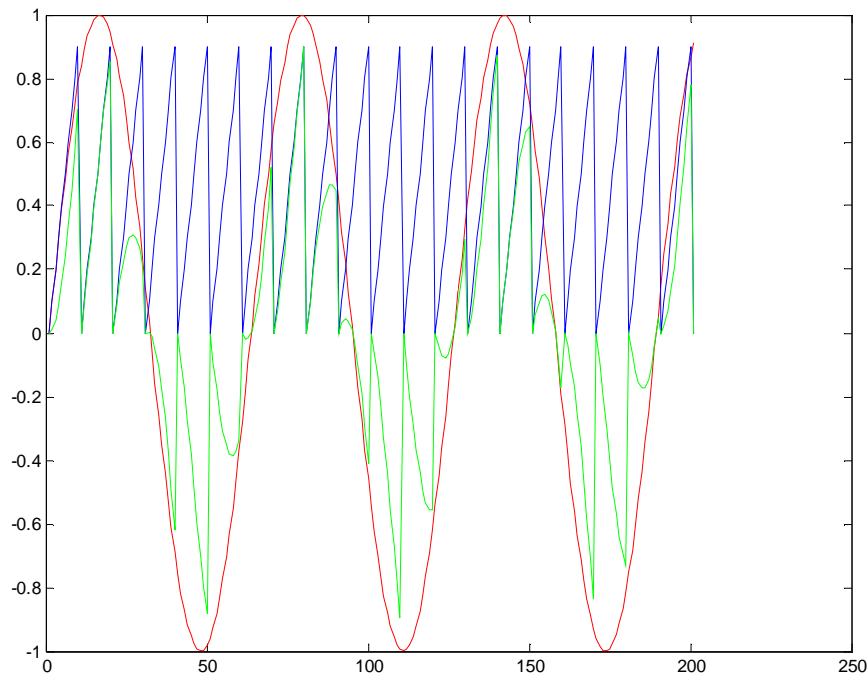


Figure 4. Modulation

- Echo: The echo effect is simply a delay played back at a different volume. The effect collects a buffer of the previous output, and plays it back at a slightly later time to give the effect of an echo.
- Chorus: The chorus effect modifies a sound so that it sounds like a number of individual versions of the sound are being played simultaneously. The sounds are modified so that the pitch and delay of a sound that is played back is altered giving the illusion of multiple active samples. This is achieved through using two delay buffers which are accessed through two oscillators (see Figure 5)

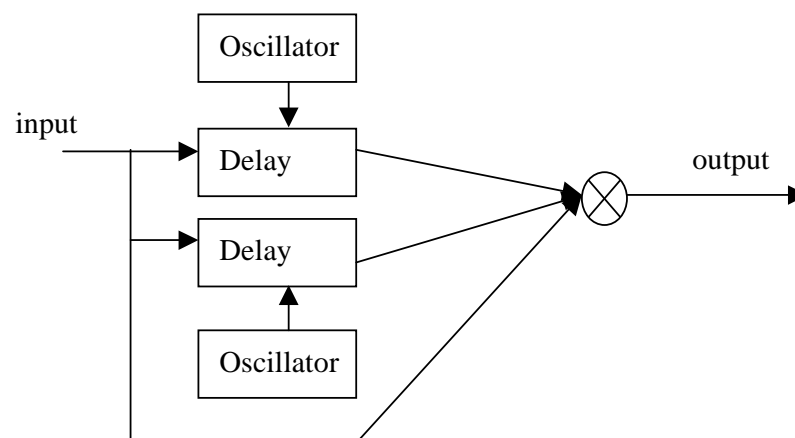


Figure 5. Chorus block diagram

Using the software synthesizer

Overview

The tiny sample generator (TSG) consists of two major sections: components and connections. An overview of the components is given in the section “*Theory – Components*”. Components can take both inputs, which determine the way they operate, and produce outputs, which produces a sound, or in turn becomes an input to another component. The way components are interconnected are described by two separate connections: Audio, and Variable (parameters).

Thus the TSG can be seen to operate in two separate phases: a variable phase, where the settings for each component are configured, and then the audio phase, where the sounds for each component are created.

Components can only be connected in a many-to-one relationship. That is, a component can accept the output from many other components, but a single component can not be the input to multiple other components. (see Figure 6.

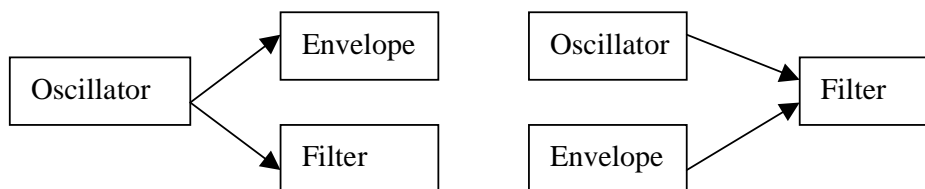


Figure 6. Relationship Diagram. (left) one-to-many [bad], (right) many-to-one [good]

Component inputs can also only flow ‘down’ within a group of components. That is, an oscillator can not be an input to an oscillator that is above it within the list.

Finally, audio data is always in the range of -1.0 to 1.0 , whereas variable data can take any range, but is generally between 0 and 10,000.

Components – Inputs & Outputs

Parameters:

Parameters have only one output and no inputs. The output is whichever value is entered. They are the simplest building block.

Oscillators:

Oscillators take a variable input and can produce both variable and audio outputs. The input variable to all oscillators is the frequency in hertz. (ie: number of times to repeat the basic pattern in Figure 1. each second). Oscillators can also be constant-based, in which case the time-value for the oscillator is replaced by a constant. For example, a sine wave oscillator normally operates as: $\sin(\text{frequency} * \text{time})$, however, under constant mode it operates as $\sin(\text{frequency} * \text{constant})$. This is useful when connecting multiple oscillators together as input devices.

Name	Oscillator
Input	Variable (Frequency (hz))
Input Range (typical)	<i>Frequency:</i> (Input variable parameter 1) Variable: 0 to 20,000 <i>Time Constant:</i> Variable: 0 to 100
Output	Variable & Audio

Offset Multiply:

Offset multiply components can take any form of input and output, as long as the type is not altered. That is, it can take an input variable, provided it produces an output variable, and an input audio provided it produces an output audio.

Name	Offset Multiply
Input	Variable & Audio (Offset, Multiply)
Input Range (typical)	<i>Offset:</i> Variables: 0 to 20,000 Audio: 0 <i>Multiply:</i> Variables: 0 to 10,000 Audio: 0 to 1 (or larger to perform distortion)
Output	Variable & Audio

Envelopes

Envelope parameters are best described by Figure 2.

Name	Envelope
Input	Audio
Input Range (typical)	<i>Start Time</i> Audio: 0 to length of sample <i>Amplitude</i> Audio: 0 to 1 <i>Attack, Decay, Sustain, Release:</i> Audio: 0 to length of sample Following the rule: Attack <= Decay <= Sustain <= Release Amplitude
Output	Audio

Filters

Each filter operates differently. A basic description of each filter is given in the theory section.

Name	Low Pass
Input	Audio & Variable (frequency, resonance)
Input Range (typical)	<i>Cut-off Frequency</i> (Input variable parameter 1) Audio: 500 to 20,000 <i>Resonance</i> : (Input variable parameter 2) (hard limit) Audio 0 to $\sqrt{2}$
Output	Audio

Name	High Pass
Input	Audio & Variable (frequency, resonance)
Input Range (typical)	<i>Cut-off Frequency</i> (Input variable parameter 1) Audio: 3000 to 20,000 <i>Resonance</i> : (Input variable parameter 2) (hard limit) Audio 0 to $\sqrt{2}$
Output	Audio

Name	Wave Shaper
Input	Audio & Variable (distortion)
Input Range (typical)	<i>Distortion</i> (Input variable parameter 1) Audio: -1.0 to 1.0 (higher to perform serious distortion)
Output	Audio

Refer to Figure 3. for wave shaper input parameter effects

Name	Smooth
Input	Audio & Variable (smoothness)
Input Range (typical)	<i>Distortion</i> (Input variable parameter 1) Audio: 0.0 to 1.0 (lower values means a smoother sound)
Output	Audio

Refer to Figure 7. below for smoothing parameter effects

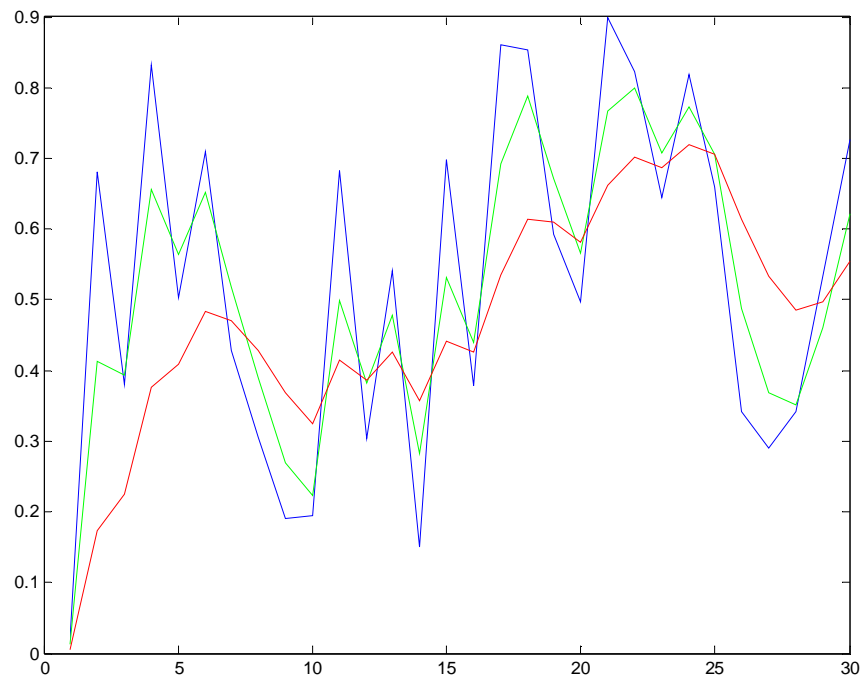


Figure 7 – Smoothing

The blue line indicates the original sample data, the green line is a smoothing of the sample with a value of 0.6, and the red line with value 0.25. Sharp changes in a line indicates what is perceived as 'clicking' noises. (as can be seen, the red (0.25) smoothing has eliminated almost all the 'clicking' noises)

Name	Modulation
Input	Audio
Input Restrictions	Modulation will only operate on two audio sources. Modulation must always have two sources.
Output	Audio

Refer to Figure 4. for modulation effects.

Name	Echo
Input	Audio, Variable (delay, amplitude)
Input Restrictions	<i>Delay</i> (Input variable parameter 1) Audio: 0 to length of sample (not recommended to be more than 1.0) <i>Amplitude</i> (Input variable parameter 2) Audio: 0 to 1
Output	Audio

Note: Amplitude indicates the amplitude of the 'echoed' signal. The original input amplitude remains unchanged

Name	Chorus
Input	Audio, Variable (frequency, frequency)
Input Restrictions	<i>Frequency</i> (Input variable parameter 1) Audio: 1 to 10 <i>Frequency</i> (Input variable parameter 2) Audio: 1 to 10 A larger value enables the perception of ‘more’ active voices
Output	Audio

Components – Allowed Connections Table

	Param	Oscil	Offset Mul	Env	Filter
Param	x	iv	iv		iv
Oscil	x	x	iv,oa,ov	oa	oa,iv
Offset Mul	x	x	x	ia,oa	ia,oa
Env	x	x	x	x	ia,oa
Filter	x	x	x		x

Table 1. Connection types

Legend:

ip – input variable
 ia – input audio
 op – output variable
 oa –output audio

Additional notes:

It is often difficult to produce the sound you want, so I either recommend you simply play with things till you make something you like the sound of, or you read up on analogue synthesis techniques and ‘patches’ so that you know how to create sounds.

Hints:

Build bandpass by connecting a highpass with lowpass
 Make strings/brass by using chorus
 Remove ‘choppy’ noises with a smoothening operation
 Use highpass on noise to make dodgy hihat
 Use lowpass on oscillator & noise to make bass drum with a kick
 Use lowpass on noise to make brown noise
 Use only the attack and decay portion of an envelope to make a ‘blast’ or ‘kick’

See this webpage for some simple schematics of percussion patches. They are generally difficult to implement because the parameters are different, and some features are not present in this synthesizer, but it gives you a general idea of what components are needed:

http://www.topher.com/analogue_percussion.html