

# Digital Audio Synthesis + Effects

## Some Practical Multimedia Digital Audio Applications

Having considered the background theory to digital audio processing, let's consider some practical multimedia related examples:

- Digital Audio Synthesis — making some sounds
- Digital Audio Effects — changing sounds via some standard effects.
- MIDI — synthesis and effect control and **compression**



Back

Close

# Digital Audio Synthesis

We have talked a lot about synthesising sounds.

Several Approaches:

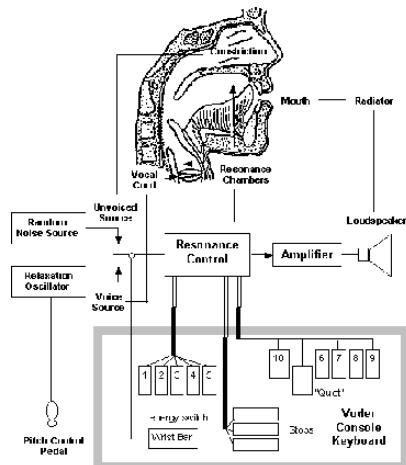
- Subtractive synthesis
- FM (Frequency Modulation) Synthesis
- Additive synthesis
- Sample-based synthesis
- Wavetable synthesis
- Granular Synthesis
- Physical Modelling

# Subtractive Synthesis

**Basic Idea:** Subtractive synthesis is a method of subtracting overtones from a sound via sound synthesis, characterised by the application of an audio filter to an audio signal.

**First Example:** Vocoder — talking robot (1939).

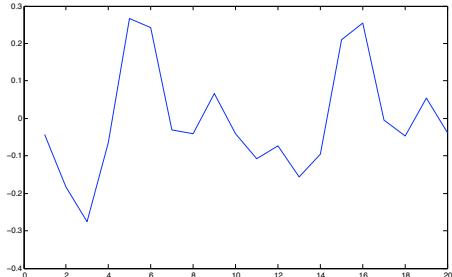
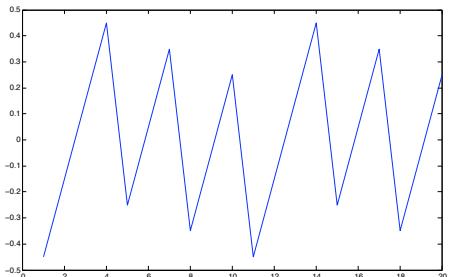
Popularised with Moog Synthesisers  
1960-1970s



# Subtractive synthesis: Simple Example

## Simulating a bowed string

- Take the output of a sawtooth generator
- Use a low-pass filter to dampen its higher partials generates a more natural approximation of a bowed string instrument than using a sawtooth generator alone.



- [subtract\\_synth.m](#) MATLAB Code Example Here.



Back

Close

# Subtractive Synthesis: A Human Example

We can regard the how humans make noises as subtractive synthesis:

**Oscillator** — the vocal cords act as the sound source and

**Filter** — the mouth and throat modify the sound.

86

- Saying or singing "ooh" and "aah" (at the same pitch.)
- Vocal chords are generating pretty much the same raw, rich in harmonic sound. Difference between the two comes from the filtering which we apply with the mouth and throat.
- Change of mouth shape varies the "cutoff frequency" of the filter, so removing (subtracting) some of the harmonics.
- The "aah" sound has most of the original harmonics still present,
- The "ooh" sound has most of them removed (or to be more precise, reduced in amplitude.)



Back

Close

# Subtractive Synthesis: Another Human Example

## A sweeping filter

"ooh"s to "aah"s again

- By gradually changing from "ooh" to "aah" and back again – simulate the "sweeping filter" effect
- Effect widely used in electronic music
- Basis of the "wahwah" guitar effect, so named for obvious reasons.
- We will see how we produce this effect in MATLAB code shortly.



Back

Close

# Subtractive Synthesis: One More Human Example

## Making Aeroplane Noise

### Make a "ssh" sound — white noise

- Now "synthesise" a "jet plane landing" sound
- Should mostly by use mouth shape to filter the white noise into pink noise by removing the higher frequencies.
- The same technique (filtered white noise) can be used to electronically synthesise the sound of ocean waves and wind,
- Used in early drum machines to create snare drum and other percussion sounds.



Back

Close

# Subtractive synthesis: Electronic Control

Three Basic elements:

**Source signal** : Common source signals on analog synths are square waves, pulse waves, sawtooth waves and triangle waves.

Modern digital and software synthesisers may include other, more complex waveforms or allow the user to upload arbitrary waveforms

**Filtering** : The cut-off frequency and resonance of the filter are controlled in order to simulate the natural timbre of a given instrument.

**Amplitude Envelope** : Further envelope control of signal amplitude (not subtractive synthesis but frequently used in practice). Also used with other synthesis techniques



Back

Close

# Further Processing: ADSR Envelope

**Basic Idea:** Modulate some aspect of the instrument's sound often its volume over time.

Why is this needed? (used by many forms of synthesis):

**When a mechanical musical instrument produces sound, the relative volume of the sound produced changes over time — The way that this varies is different from instrument to instrument**

## Examples:

**Pipe Organ** : when a key is pressed, it plays a note at constant volume; the sound dies quickly when the key is released.

**Guitar** : The sound of a guitar is loudest immediately after it is played, and fades with time.

Other instruments have their own characteristic volume patterns.

**Also Note:** While envelopes are most often applied to volume, they are also commonly used to control other sound elements, such as filter frequencies or oscillator pitches.



Back

Close

# Further Processing: ADSR Envelope (Cont.)

**Attack** : How quickly the sound reaches full volume after the sound is activated (the key is pressed).

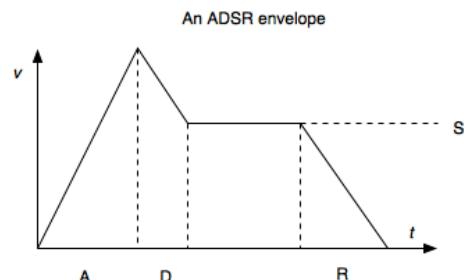
- For most mechanical instruments, this period is virtually instantaneous.
- For bowed strings or some popular synthesised "voices" that don't mimic real instruments, this parameter is slowed down. 'Slow attack' is commonly part of sounds — 'pads'.

**Decay** : How quickly the sound drops to the sustain level after the initial peak.

**Sustain** : The "constant" volume that the sound takes after decay until the note is released. Note that this parameter specifies a volume level rather than a time period.

**Release** How quickly the sound fades when a note ends (the key is released).

- Often, this time is very short. e.g. organ
- An example where the release is longer might be a bell ring, or a piano with the sustain pedal pressed.



Back

Close

# Using MATLAB Filter Example: Subtractive Synthesis Example

The example for studying subtractive synthesis, subtract\_synth.m, uses the butter and filter MATLAB functions:

```
% simple low pas filter example of subtractive synthesis
Fs = 22050;
y = synth(440,2,0.9,22050,'saw');

% play sawtooth e.g. waveform
doit = input('\nPlay Raw Sawtooth? Y/[N]:\n\n', 's');
if doit == 'y',
    figure(1)
    plot(y(1:440));
    playsound(y,Fs);
end

%make lowpass filter and filter y
[B, A] = butter(1,0.04, 'low');
yf = filter(B,A,y);

[B, A] = butter(4,0.04, 'low');
yf2 = filter(B,A,y);

% play filtererd sawtooths
doit = ...
```



Back

Close

```
input('\nPlay Low Pass Filtered (Low order) ? Y/[N]:\n\n', 's');
if doit == 'y',
figure(2)
plot(yf(1:440));
playsound(yf,Fs);
end

doit = ...
input('\nPlay Low Pass Filtered (Higher order)? Y/[N]:\n\n', 's');
if doit == 'y',
    figure(3)
plot(yf2(1:440));
playsound(yf2,Fs);
end

%plot figures
doit = input('\nPlot All Figures? Y/[N]:\n\n', 's');
if doit == 'y',
figure(4)
plot(y(1:440));
hold on
plot(yf(1:440),'r+');
plot(yf2(1:440),'g-');
end
```



Back

Close

## synth.m

The supporting function, synth.m, generates waveforms as we have seen earlier in this tutorial:

```
function y=synth(freq,dur,amp,Fs,type)
% y=synth(freq,dur,amp,Fs,type)
%
% Synthesize a single note
%
% Inputs:
% freq - frequency in Hz
% dur - duration in seconds
% amp - Amplitude in range [0,1]
% Fs - sampling frequency in Hz
% type - string to select synthesis type
%         current options: 'fm', 'sine', or 'saw'

if nargin<5
    error('Five arguments required for synth()');
end

N = floor(dur*Fs);
n=0:N-1;
if (strcmp(type,'sine'))
    y = amp.*sin(2*pi*n*freq/Fs);
```



```
elseif (strcmp(type,'saw'))  
  
T = (1/freq)*Fs; % period in fractional samples  
ramp = (0:(N-1))/T;  
y = ramp-fix(ramp);  
y = amp.*y;  
y = y - mean(y);  
  
elseif (strcmp(type,'fm'))  
  
t = 0:(1/Fs):dur;  
envel = interp1([0 dur/6 dur/3 dur/5 dur], [0 1 .75 .6 0], ...  
    0:(1/Fs):dur);  
I_env = 5.*envel;  
y = envel.*sin(2.*pi.*freq.*t + I_env.*sin(2.*pi.*freq.*t));  
  
else  
    error('Unknown synthesis type');  
end  
% smooth edges w/ 10ms ramp  
if (dur > .02)  
    L = 2*fix(.01*Fs)+1; % L odd  
    ramp = bartlett(L)'; % odd length  
    L = ceil(L/2);  
    y(1:L) = y(1:L) .* ramp(1:L);  
    y(end-L+1:end) = y(end-L+1:end) .* ramp(end-L+1:end);  
end
```

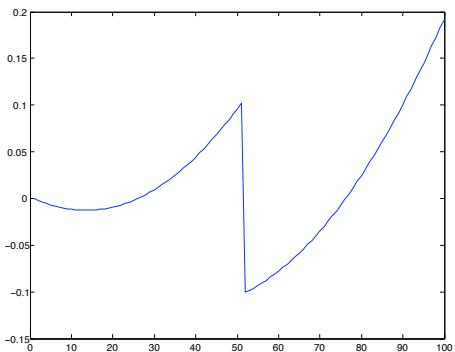


Back

Close

## synth.m (Cont.)

Note the *sawtooth* waveform generated here has a non-linear up slope:



This is created with:

```
ramp = (0 : (N-1)) / T;  
y = ramp-fix(ramp);
```

`fix` rounds the elements of `X` to the nearest integers towards zero.

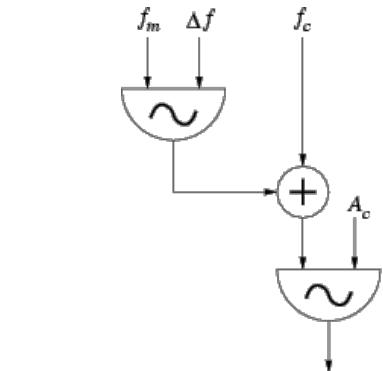
This form of sawtooth sounds slightly less harsh and is more suitable for audio synthesis purposes.

# FM (Frequency Modulation) Synthesis

**Basic Idea:** Timbre of a simple waveform is changed by frequency modulating it with a frequency resulting in a more complex waveform — different-sounding.

**Discovered** by John Chowning at Stanford University in 1967-68,  
**Patented in 1975** and was later licensed to Yamaha.

**Used** in popular 1980s Yamaha Synthesisers: DX7, Casio CZ ..... **still in use today**



# FM (Frequency Modulation) Synthesis

## (cont.)

- Radio broadcasts use FM in a different way
- FM synthesis is very good at creating both harmonic and inharmonic ('clang', 'twang' or 'bong' noises) sounds
  - For synthesizing harmonic sounds, the modulating signal must have a harmonic relationship to the original carrier signal.
  - As the amount of frequency modulation increases, the sound grows progressively more complex.
  - Through the use of modulators with frequencies that are non-integer multiples of the carrier signal (i.e., non harmonic), bell-like dissonant and percussive sounds can easily be created.

# FM (Frequency Modulation) Synthesis (cont.)

- Digital implementation — true analog oscillators difficult to use due to instability
- 1960s origin analog – FM discovered when vibrato sped up to the point that it was creating audible sidebands (perceived as a timbral change) rather than faster warbling (perceived as a frequency change).
- **DX synthesiser FM** - Where both oscillators use Sine waves and are "musically-tuned" frequencies generated from a keyboard



# FM Synthesis: Mathematical Underpinnings Definitions

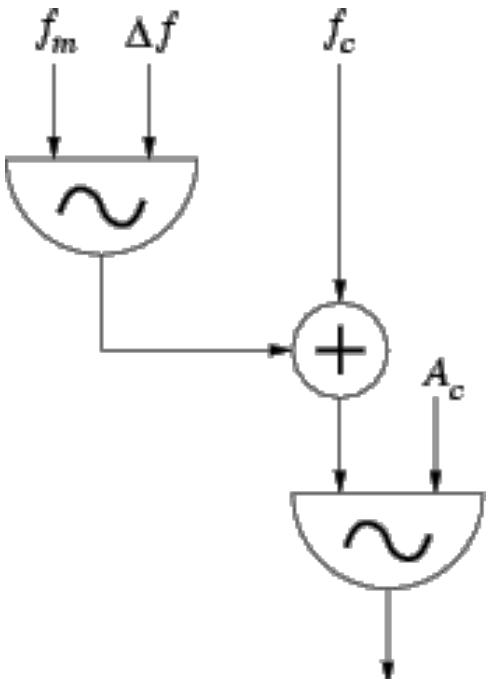
**Oscillator** : A device for generating waveforms

**FM Frequency (Pitch) Modulation**

: Where the frequency (pitch) of an oscillator (*the Carrier*) is modulated by another oscillator (*the Modulator*)

**Carrier Frequency** : The frequency of the oscillator which is being modulated

**Modulator Frequency** : The frequency of the oscillator which modulates the Carrier



# FM Synthesis: Basic Frequency Modulation

## Basic FM Equation:

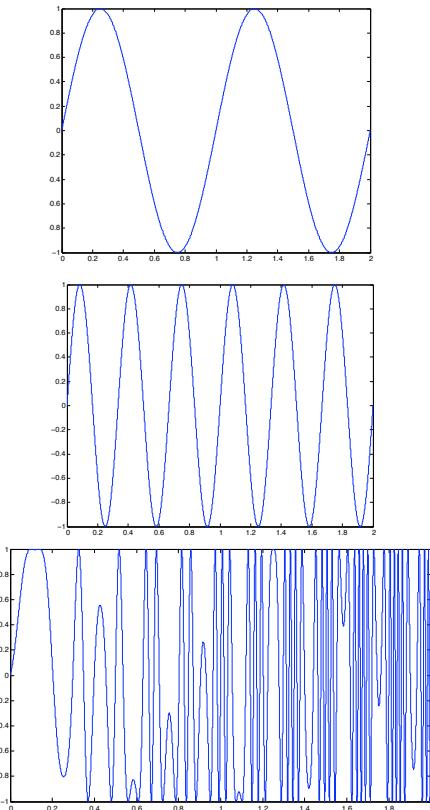
$$e = A \sin(\alpha t + I \sin \beta t)$$

$A$  is the peak amplitude

$e$  is the instantaneous amplitude of the modulated carrier

$\alpha$  and  $\beta$  are the respective carrier and modulator frequencies

$I$  is the modulation index: the ratio of peak deviation to modulator frequency



# MATLAB FM Example

MATLAB code to produce basic FM ([fm\\_eg.m](#),  
see also [fm\\_eg\\_plot.m](#)):

```
% Signal parameters
fs = 22050;
T = 1/fs;
dur = 2.0;          % seconds
t = 0:T:dur;       % time vector

% FM parameters
fc = 440;          % center freq
fm = 30;
Imin = 0;
Imax = 20;
I = t.* (Imax - Imin)/dur + Imin;

y = sin(2*pi*fc*t + I.*sin(2*pi*fm*t));
plot(t(1:10000), y(1:10000));

playsound(y, fs);
```



Back

Close

# FM Synthesis: Side Frequencies

The harmonic distribution of a simple sine wave signal modulated by another sine wave signal can be represented with Bessel functions

$$\begin{aligned} e = & A\{J_0 \sin \alpha t \\ & + J_1 [\sin(\alpha + \beta)t - \sin(\alpha - \beta)t] \\ & + J_2 [\sin(\alpha + 2\beta)t - \sin(\alpha - 2\beta)t] \\ & + J_3 [\sin(\alpha + 3\beta)t - \sin(\alpha - 3\beta)t] \\ & \dots \} \end{aligned}$$

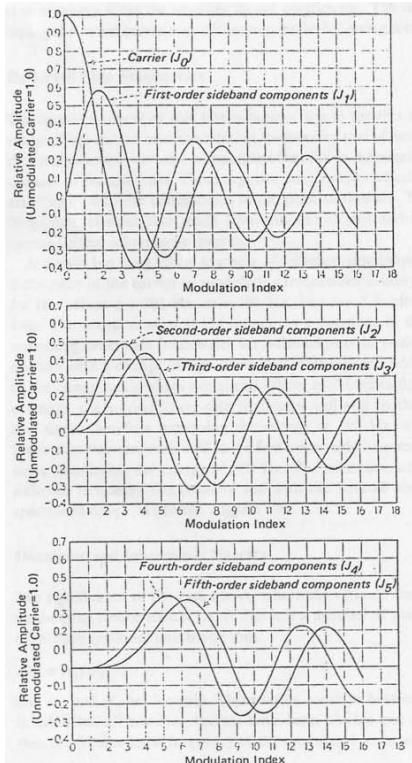
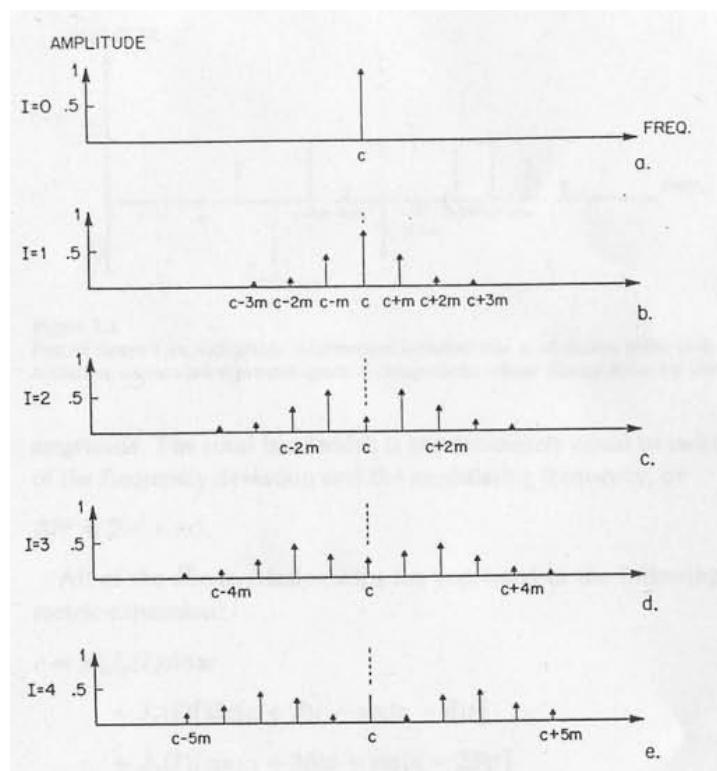
- Provides a basis for a simple mathematical understanding of FM synthesis.
- **Side Frequencies** produced and are related to modulation index,  $I$ 
  - If  $I > 1$  energy is *increasingly stolen* from the carrier but with constant modulation frequency.



Back

Close

# FM Synthesis: Side Frequencies (Cont.)



# FM Synthesis: Making Complex Sounds

## Operators and Algorithms

**Operators** are just Oscillators in FM Terminology.

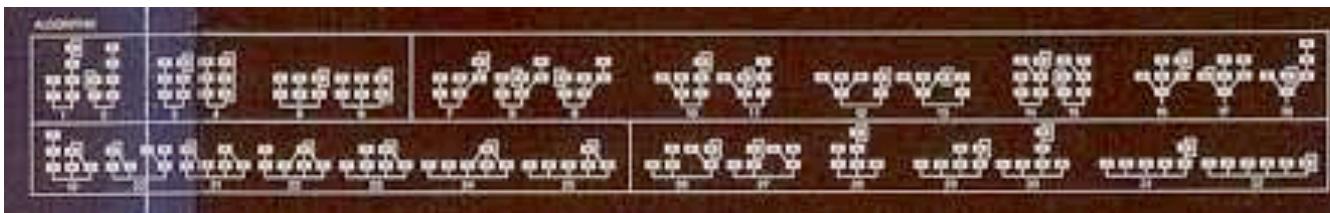
- FM synths will have either 4 or 6 Operators.
- **Why so many Operators?**

Multimedia  
CM0340

105

Sounds from one Modulator and one Carrier aren't exactly that overwhelmingly complex

**Algorithms** are the preset combinations of routing available to you.



&lt;&lt;

&gt;&gt;

&lt;

&gt;

Back

Close

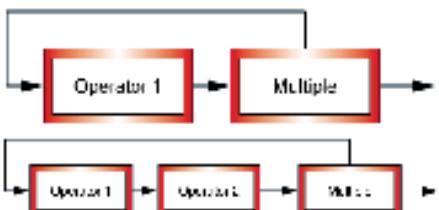
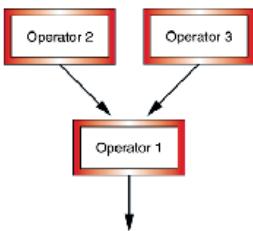
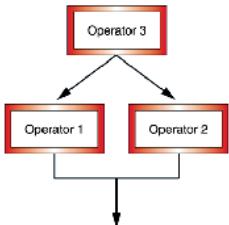
# FM Synthesis: FM Algorithms

## How to connect up Operators?

**Multiple Carriers** : One oscillator simultaneously modulates two or more carriers

**Multiple Modulators** : Two or more oscillators simultaneously modulate a single carrier

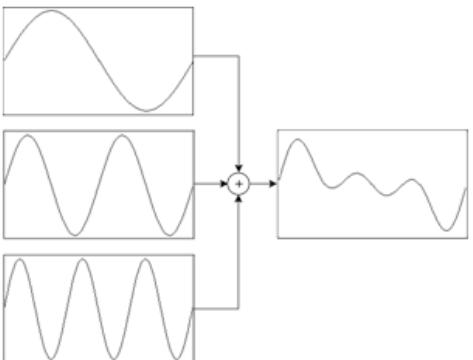
**Feedback** : Output of oscillator modulates the same oscillator



# Additive synthesis

**Basic Idea:** Additive synthesis refers to the idea that complex tones can be created by the summation, or addition, of simpler ones.

- Frequency *mixing* is the essence of additive synthesis.
- Each of the frequency components (or partials) of a sound has its own amplitude envelope.
- This allows for independent behaviour of these components.
- Sources can be other forms of synthesis or samples.



# Additive synthesis: Examples

**Organs** : Pipe organs or Hammond organs.

The concept of register-stops of organs = additive synthesis:

- complex timbres result from the addition of different components to the spectrum of a sound.
- Different pipe stops or tonewheel/drawbar settings



**Telharmonium** : An early giant electrical synthesiser (1900s):

- adds together the sounds from dozens of electro-mechanical tone generators to form complex tones.
- Important place in the history of electronic and computer music.

**Modern Variants** : Fairlight CMI, Synclavier, Kawai K5000 series, wavetable synthesis (more soon)

# Additive synthesis: Basic Theory

- Basis: Fourier Theory
- Simply stated: a complex timbre that has been analysed into its sinusoidal components can then be reconstructed by means of additive synthesis.
- Additive synthesis has the advantage that the many micro-variations in the frequency and amplitude of individual partials, that make natural sounds so rich and lively, can be recreated.
- The disadvantage with this form of synthesis is its inefficiency in that a great deal of data must be specified to define a sound of any complexity of detail.
- Simple MATLAB Example: [additive\\_synth.m](#)



Back

Close

# Sample-based synthesis

**Basic Ideas:** Similar to either subtractive synthesis or additive synthesis.

The **principal difference** is that the seed waveforms are sampled sounds or instruments instead of fundamental waveforms such as the saw waves of subtractive synthesis or the sine waves of additive synthesis.

Samplers, together with traditional Foley artists, are the mainstay of modern sound effects production.

Musical genres: Hip-hop, Trip-hop, Dance music, Garage, Jungle, Trance, Modern Electronica *invented* due to samplers.

Most music production now uses samplers.



Back

Close

# Sample-based synthesis: Comparison with other Synthesis methods

- Advantages (over other methods of digital synthesis such as physical modelling synthesis (**more soon**) or additive synthesis): processing power requirements are much lower.
  - Nuances of the sound models are contained in the pre-recorded samples rather than calculated in real-time.
- Disadvantage: in order to include more detail, multiple samples might need to be played back at once
  - E.g. a trumpet might include a breath noise, a growl, and a looping soundwave used for continuous play
  - Reduces the polyphony as sample-based synthesizers rate their polyphony based on the number of multi-samples that can be played back simultaneously.

# Sample-based synthesis: Examples

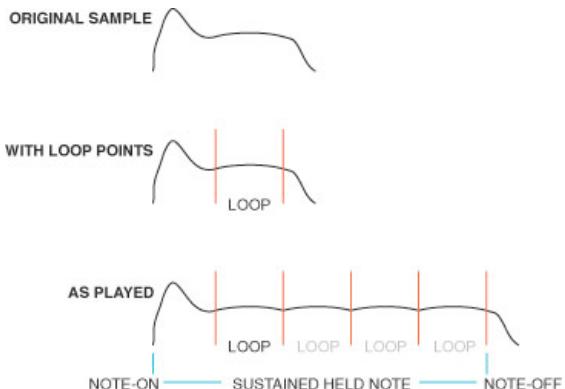
- Mellotron (analog tape) (1962)
- Computer Music Melodian (1976): Stevie Wonder's "Secret Life of Plants"
- CMI Fairlight (1979)
- NED Synclavier (1979).
- EMU Emulator series (1981)
- Akai S Series (1986)
- Korg M1 (1988): The M1 also introduced the "workstation" concept.
- Software Samplers (2005) : NI Kontakt, Steinberg Halion



# Sample-based synthesis Basics: Looping

A sample-based synthesizer's ability to reproduce the nuances of natural instruments is determined primarily by its library of sampled sounds.

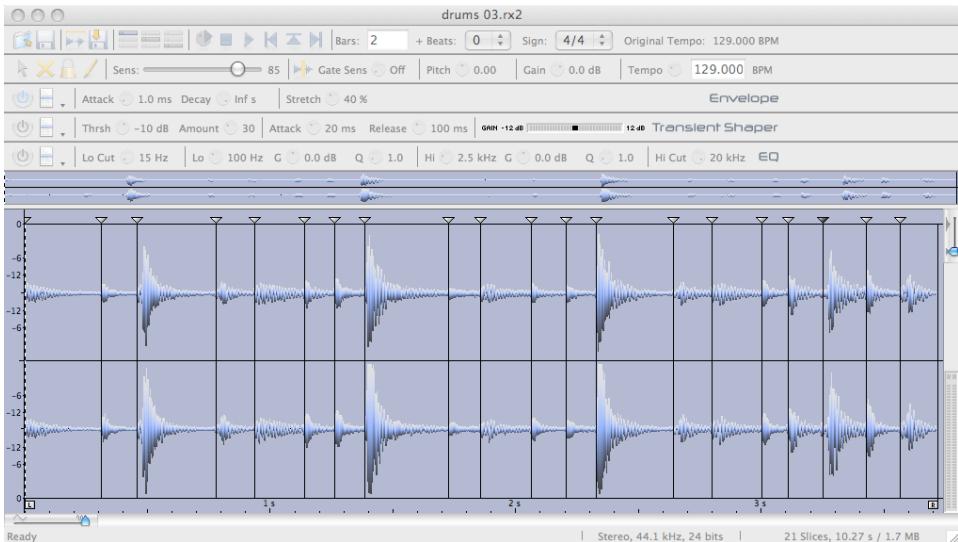
- Early days: computer memory expensive:  
Samples had to be as **short** and as **few** as possible.  
This was achieved by **looping a part of the sample**



- Problem: How to find looping points?

# Finding looping points

- Simple idea: Find silence points (**zero (amplitude) crossings**) in sample.  
E.g. Drum beats



Loop between these

- Alternative: Find portions in sample that have same audio content — pattern matching.  
E.g. Sustaining musical instruments.

# Sample-based synthesis Basics: Looping (cont)

## Pitch control:

- Speed or slow up sample to change pitch (realism to only a few semitones in pitch change)
- Still need some sample across the range of the keyboard
- As memory became cheaper (and now with software based sample synthesis), it became possible to use **multisampling** — looping still used in individual samples.

## Finishing off the loop:

- Early Days: Use a volume envelope curve to make the sound fade away.
- Today: Include tail off sample in data — triggered by note off.



Back

Close

# Beat Slicing Algorithms Background: Altering Loops

**Silence Points** : Find silence points (**zero (amplitude) crossings**) in sample.

Snapping to silence points means that no nasty clicks in audio when joining audio together. (Too simple)

**Beat Perception** : *The human listening system determines the rhythm of music by detecting a pseudo periodical succession of beats*

- The more energy the sound transports, the louder the sound will seem.
- But a sound will be heard as a beat only if his energy is largely superior to the sounds energy history, that is to say if the brain detects a large variation in sound energy.
- Therefore if the ear intercepts a monotonous sound with sometimes big energy peaks it will detect beats,



# Beat Slicing Algorithms Ideas (1):

Simple sound energy :

- Computing the **average** sound energy of the signal over a relatively large sample (around 1 second)
- Compute **instant** sound energy (around 5/100 second).
- Comparing average to the instant sound energy.
- We detect a beat only when the instant energy is larger than the local energy average.



Back

Close

# Beat Slicing Algorithms Ideas (2):

Frequency selected sound energy : More elaborate model:

*Try to detect big sound energy variations in particular frequency subbands*

- Apply Fourier Transform — separate beats according to their frequency sub-band.
- Apply energy analysis but in frequency space:
  - Compute Fourier Transform over 1024 samples.
  - Divide into around 32 sub-bands.
  - Compute the sound energy contained in each of the subbands
  - Compare it to the recent energy average corresponding to this subband.
  - If one or more subbands have an energy superior to their average we have detected a beat.
- For more details search web or see references at end of slides.

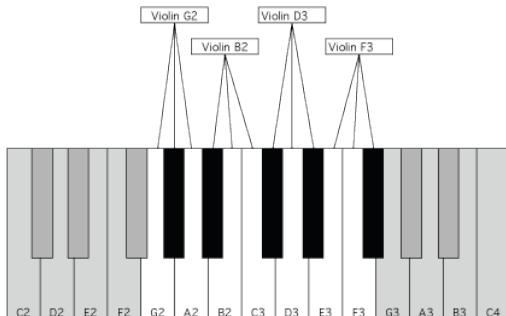


Back

Close

# Sample-based Synthesis: Multisampling

- Non-pitched simple example: the concept of drum mapping —  
**see also general MIDI section later**
- **Need to preserve relationships between key notes**
- **Multisampling Basic Idea:**
  - Sample instrument at regular intervals to cover regions of several adjacent notes (*splits*) or for every note.
  - **Advantage:** provides a more natural progression from the lower to the higher registers



# Sample-based Synthesis: Example Kontakt Sampler Multisample Keymap



◀◀

▶▶

◀

▶

Back

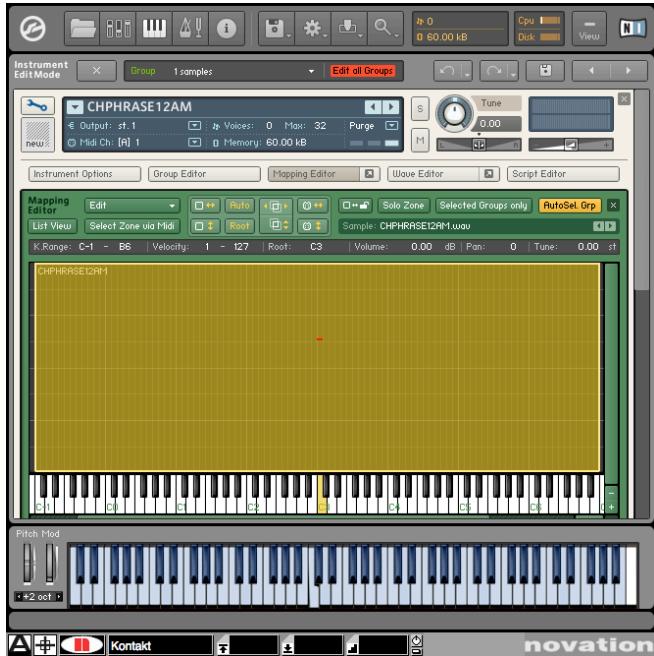
Close

# Sample-based Synthesis: Velocity Layers

- When pluck a string or hit a drum or press a piano key, sound produced **depends** on how hard the action was.
- In software, this is measured by the velocity of a key press *etc.*
- Multisampling lays out samples vertically in keymap.
- **Velocity layers** layed out **horizontally**



# Sample-based Synthesis: Example Velocity Layers (1)



(Single key mapped) Single Velocity Layer — Only one type of sound played at any velocity.

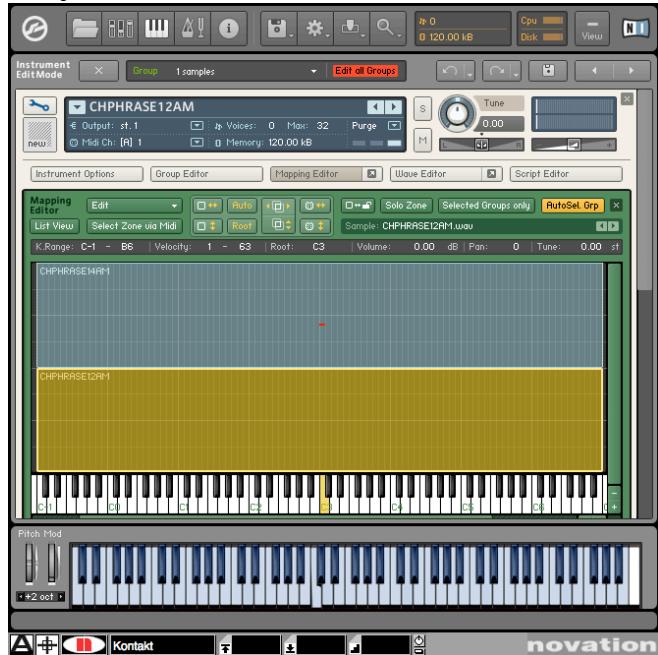
Volume output maybe controlled by velocity but not change in timbre of sound.

# Sample-based Synthesis: Example Velocity Layers (2)

(Single key mapped) Dual Velocity Layer:

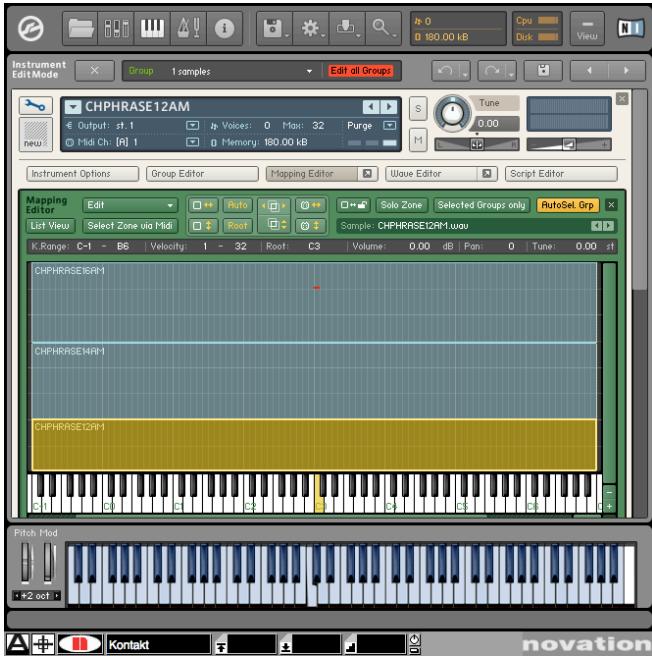


Sound one played at lower level velocity



Sound Two played at higher velocity

# Sample-based Synthesis: Example Velocity Layers (3)



(Single key mapped) Triple Velocity Layer — Three type of sounds played according to velocity.

Here upper velocity sound is being played.

# Sample-based Synthesis: Key Map and Velocity Layers



Most instruments are a combination of multisample key mapped and velocity layers

# Sample-based synthesis Basics: Sample Keyswitching

- Instruments can make vastly different sounds depending how they are played
- Example: Trumpets (muted/not muted), violin (plucked, slow/fast up/down bow)
- For expressive performance samples can be **keyswitched**:  
Use keys (usually lower keys outside of instrument range) to select appropriate sounds
  - Essentially banks of keymapped velocity layered samples



G = Bucket mute  
F = Harmon mute  
E = Cup mute  
D = Straight mute  
C = Open (no mute)

Example: If you press the "F" key the instrument will switch to a Harmon mute.



# Advanced Software Based Sampling

- Sampling now seems to have very few limits
- Full orchestras and even choirs that can sing
- Can sing words too (Advanced Keyswitching).
- Programming script control over sampler (Kontakt 2 and above).



# Wavetable synthesis

## What is Wavetable music synthesis?

Similar to simple digital sine wave generation/additive synthesis but extended at least two ways.

- **Waveform lookup table** contains samples for not just a single period of a sine function but for a single period of a **more general waveshape**.
- Mechanisms exists for dynamically changing the waveshape as the musical note evolves:  
thus generating a quasi-periodic function in time.

**Not to be confused with common PCM sample buffer playback: soundcards**

# Wavetable synthesis: Examples

**PPG Wave Series:** Implementation of wavetable synthesis employed an array containing 64 pointers to individual single-cycle waves.

**Waldorf Microwave:** Next generation PPG.

**Roland D-50 and Roland MT-32/variants:**  
 "Linear Arithmetic" synthesizers — combined complex sampled attack phases with less complex sustain/decay phases (basically a wavetable synthesizer with a 2-entry wave sequence table).

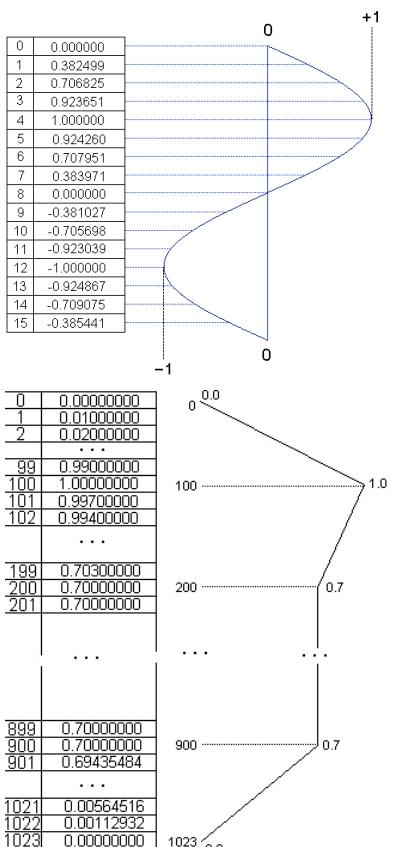
**Sequential Circuits Prophet-VS,**

**Korg Wavestation:** "Vector synthesis" — move through wavetables and sequences arranged on a 2-dimensional grid.



# Wavetable Basics: Making Waves

- The sound of an existing instrument (a single note) is sampled and parsed into a circular sequence of samples or wavetables:
  - each having one period or cycle per wave;
  - A set of wavetables with user specified harmonic content can also be generated mathematically.
- At playback, these wavetables are used to fetch samples (table-lookup)
- However** the output waveform is not normally static and evolves slowly in time as one wavetable is mixed with another, creating a changing waveform via [ADSR Enveloping](#).
- Looping maybe used to slow, reverse wavetable evolution



# Wavetable Basics: Practicalities

Put more simply, a wavetable synthesiser will store **two parts** of an instrument's sound.

- A sample of the attack section (e.g. the sound of the hammer hitting a piano string)
- A small segment of the sustain portion of the instrument's sound.

When triggered:

- The **attack** sample is played once immediately followed by a loop of the sustain segment.
- The endlessly looping segment is then enveloped to create a natural sounding decay (dying away of the sound).



Back

Close

# Wavetable v. Sample Playback

- **Differs** from simple sample playback as
  - Output waveform is always generated in real time as the CPU processes the wave sequences
  - Waves in the tables are rarely more than 1 or 2 periods in length.



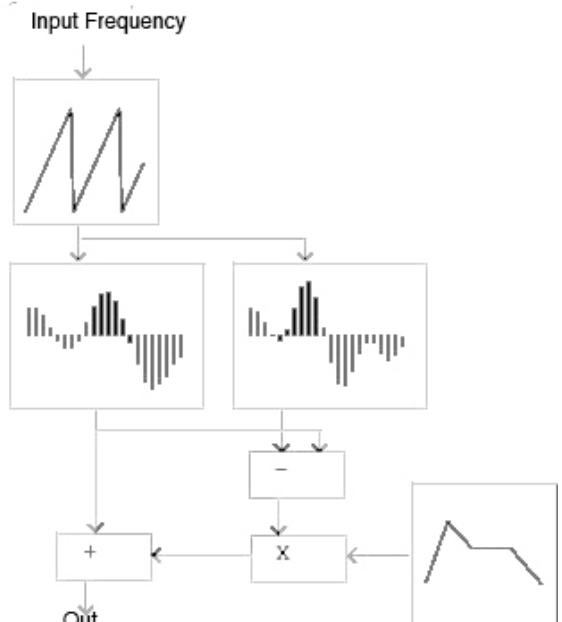
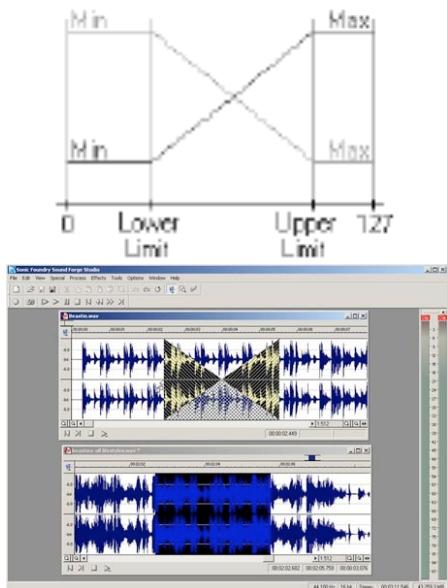
Back

Close

# Wavetable synthesis: Dynamic Waveshaping (1)

Simplest idea: **Linear crossfading**

- Crossfade from one wavetable to the next sequentially.
- Crossfade = apply some envelope to smoothly merge waveforms.



# Wavetable Synthesis Examples

Simple example — create one sine wave and one saw and then some simple cross-fading between the waves: wavetable\_synth.m.

```
% simple example of Wavetable synthesis

f1 = 440; f2 = 500; f3 = 620;
Fs = 22050;

%Create a single sine waves of frequencie f1
y1 = synth(f1,1/f1,0.9,Fs,'sine');

doit = input('\nPlay/Plot Raw Sine y1 looped for 10 ...
seconds? Y/[N]:\n\n', 's');
if doit == 'y',
figure(1)
plot(y1);
loopsound(y1,Fs,10*Fs/f1);
end

%Create a single Saw waves of frequencie f2
y2 = synth(f2,1/f2,0.9,Fs,'saw');

doit = input('\nPlay/Plot Raw saw y2 looped for 10 ...
```



Back

Close

```
seconds? Y/[N]:\n\n', 's');
if doit == 'y',
figure(2)
plot(y2);
loopsound(y2,Fs,10*Fs/f2);
end

%concatenate wave
ywave = [y1 , y2];

% Create Cross fade half width of wave y1 for xfade window
xfadewidth = floor(Fs/(f1*2));
ramp1 = (0:xfadewidth)/xfadewidth;
ramp2 = 1 - ramp1;

doit = input('\nShow Crossfade Y/[N]:\n\n', 's');
if doit == 'y',
figure(4)
plot(ramp1);
hold on;
plot(ramp2,'r');
end;

%apply crossfade centered over the join of y1 and y2
pad = (Fs/f1) + (Fs/f2) - 2.5*xfadewidth;
xramp1 = [ones(1,1.5*xfadewidth) , ramp2, zeros(1,pad)];
xramp2 = [zeros(1,1.5*xfadewidth) , ramp1, ones(1,pad)];
```



Back

Close

```
% Create two period waveforms to fade between
ywave2 = [y1 , zeros(1,Fs/f2)];;
ytemp = [zeros(1,Fs/f1), y2];

ywave = ywave2;
% do xfade

% add two waves together over the period modulated by xfade ramps
% (recall .* to multiply matrices element by element
% NOT MATRIX multiplication

ywave2 = xramp1.*ywave2 + xramp2.*ytemp;

doit = input ('\nPlay/Plot Additive Sines together? Y/[N]:\n\n', 's');
if doit == 'y',
figure(5)

subplot(4,1,1);
plot(ywave);

hold off

set(gca,'fontsize',18);
ylabel('Amplitude');
title('Wave 1');
set(gca,'fontsize',18);
```



Back

Close

```
subplot(4,1,2);
plot(ytemp);
set(gca,'fontsize',18);
ylabel('Amplitude');
title('Wave 2');
set(gca,'fontsize',18);
subplot(4,1,3);
plot(xramp1);
hold on
plot(xramp2,'r')
hold off
set(gca,'fontsize',18);
ylabel('Amplitude');
title('Crossfade Masks');
set(gca,'fontsize',18);
subplot(4,1,4);
plot(ywave2);
set(gca,'fontsize',18);
ylabel('Amplitude');
title('WaveTable Synthesis');
set(gca,'fontsize',18);
loopsound(ywave2,Fs,10*Fs/(f1 + f2));
end
```



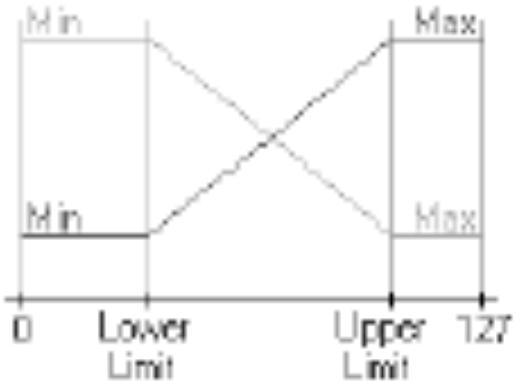
Back

Close

# Wavetable synthesis: Dynamic Waveshaping (1)

Simplest idea: [Linear crossfading](#)

- Crossfade from one wavetable to the next sequentially.
- Crossfade = apply some envelope to smoothly merge waveforms.



# Simple MATLAB Example: Linear Crossfading

Simple MATLAB wavetable/crossfading example:

wavetable\_synth.m:

```
% Create Cross fade half width of wave y1 for xfade window
xfadewidth = floor(Fs/(f1*2));
ramp1 = (0:xfadewidth)/xfadewidth;
ramp2 = 1 - ramp1;

%apply crossfade centered over the join of y1 and y2
pad = (Fs/f1) + (Fs/f2) - 2.5*xfadewidth;
xramp1 = [ones(1,1.5*xfadewidth) , ramp2, zeros(1,pad)];
xramp2 = [zeros(1,1.5*xfadewidth) , ramp1, ones(1,pad)];
% Create two period waveforms to fade between
ywave2 = [y1 , zeros(1,Fs/f2)];
ytemp = [zeros(1,Fs/f1), y2];

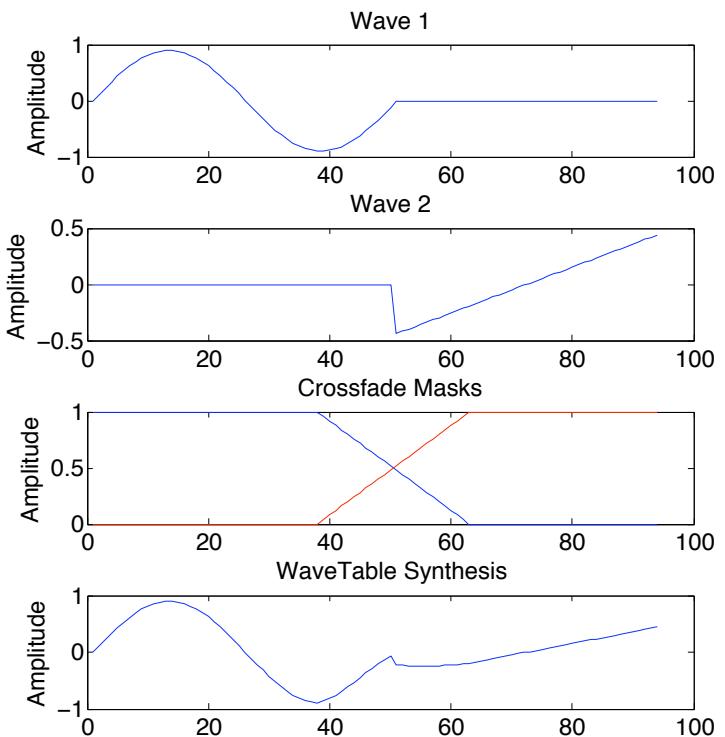
% do xfade
% add two waves together over the period modulated by xfade ramps
ywave2 = xramp1.*ywave2 + xramp2.*ytemp;
```



Back

Close

# Simple MATLAB Example: Linear Crossfading (Cont.)

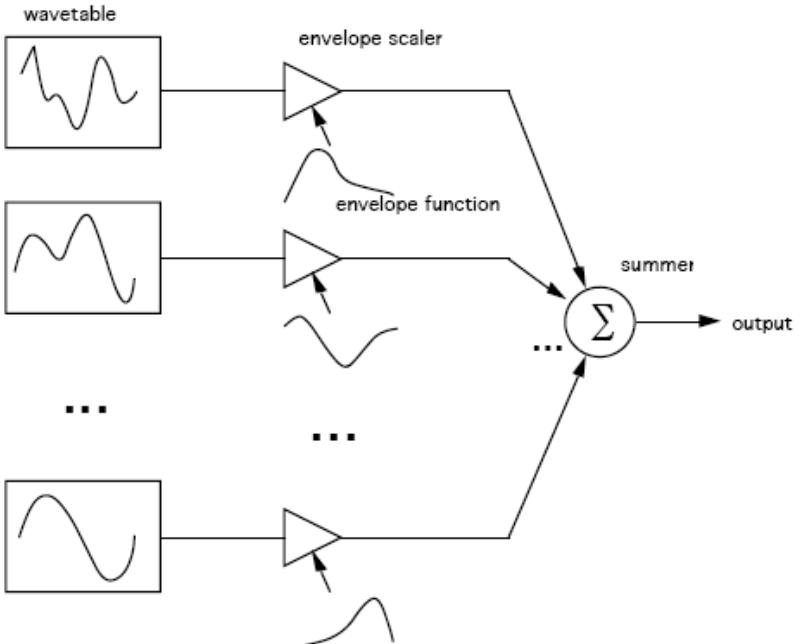


Note: This sort of technique is useful to create an ADSR envelope in MATLAB

# Wavetable synthesis: Dynamic Waveshaping (2)

More sophisticated method: **Sequential Enveloping**

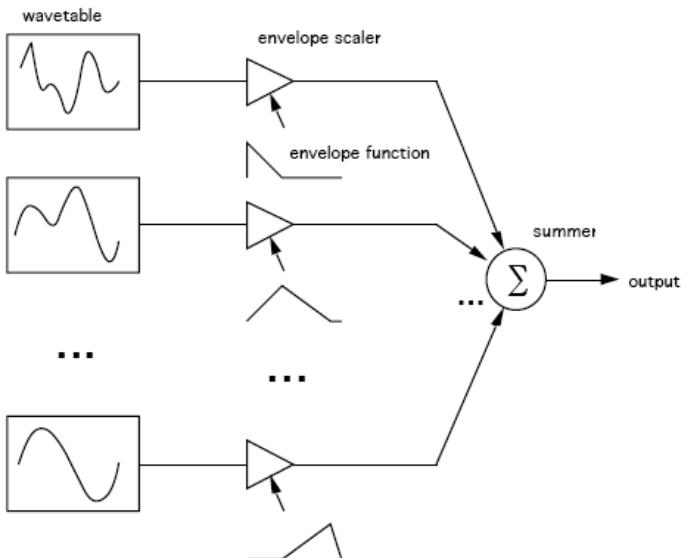
- Example below: two wavetables are being mixed at any one instance of time by moving envelope scale



# Wavetable synthesis: Dynamic Waveshaping (3)

## Linear Crossfading as Sequential Enveloping?

- The simple linear crossfading method can be thought of as a subclass of the more general basis mixing method where the envelopes are overlapping triangular pulse functions.



# Wavetable synthesis: Advantages

- Well suited for synthesizing quasi-periodic musical tones because wavetable synthesis can be as compact in storage requirements
  - Amount of data being stored and used for this synthesis method is far less than just the PCM sample of same sound.
  - As general as additive synthesis but requires much less real-time computation.
- Wavetable synthesis takes advantage of the quasiperiodic nature of the waveform to remove redundancies and to reduce the data.  
A bit more technical:
  - Precomputes the inverse Discrete Fourier Transform (DFT) of the waveform spectrum before playback
  - Rather than computing the inverse DFT in real-time as additive synthesis does.
  - Precomputed, real-time synthesis is reasonably simple to implement.



Back

Close

# Granular Synthesis

*"All sound is an integration of grains, of elementary sonic particles, of sonic quanta."* -Iannis Xenakis, Greek Composer (1971).

Granular synthesis:

- Sound synthesis method that operates on the microsound time scale.
- Based on the same principles as sampling/wavetable synthesis but often includes analog technology as well.
- **Difference** Samples are not used directly to make usual sounds:
  - Split in small pieces of around 1 to 50 ms (milliseconds) in length, **the grains**.
  - Multiple grains may be layered on top of each other all playing at different speed, phase and volume.



Back

Close

# Granular Synthesis: Soundscape

Result is no single tone, but a soundscape!

- Often a cloud, that is subject to manipulation
- Unlike any natural sound and also unlike the sounds produced by most other synthesis techniques.
- By varying the waveform, envelope, duration, spatial position, and density of the grains many different sounds can be produced.

# Granular Synthesis: Is this music?

- Usable as music or soundscapes (ambient)
- Usable as Sound effects
- Usable to alter sample speed while preserving the original pitch/tempo information —**pitch/tempo synchronous granular synthesis**
- Usable as Raw material for further processing by other synthesis or DSP effects.
- The range of effects that can be produced include amplitude modulation, time stretching, stereo or multichannel scattering, random reordering, disintegration and morphing.



Back

Close

# Granular Synthesis: Background

Strong Physics Background:

- Quantum physics has shown that sound can be atomically reduced to physical particles
- Physical form of sound was first envisioned by the Dutch scientist Isaac Beeckman (1618):  
*"Sound travels through the air as globules of sonic data."*
- Denis Gabor (1947) proposed the idea of a grain as the quantum of sound and more recently
- Xenakis (1971) first musical use of granular synthesis — a reel to reel tape recorder, a razor blade, sticky tape, and a lot of time.
- Curtis Roads (1988), digital granular synthesis
- Barry Truax (1990) real-time granular synthesis composition  
Riverrun, Buy the CD!



Back

Close

# Granular Synthesis: Implementations

- Software:

Many implementations nowadays:

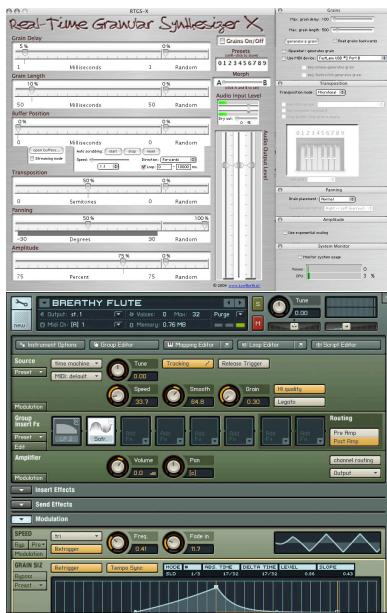
**Programmable :** Csound,  
MATLAB,  
routines:  
MAX/MSP

**Standalone :** SuperCollider,  
Granulab, RTGS X.

**DAW plug-ins standalone :**  
VSTis etc.

**Modern Music Samplers :**  
Native Instruments' Kontakt,  
Intakt...., others

- Hardware: Korg Kaos Pad.



# Granular Synthesis: What is a grain?

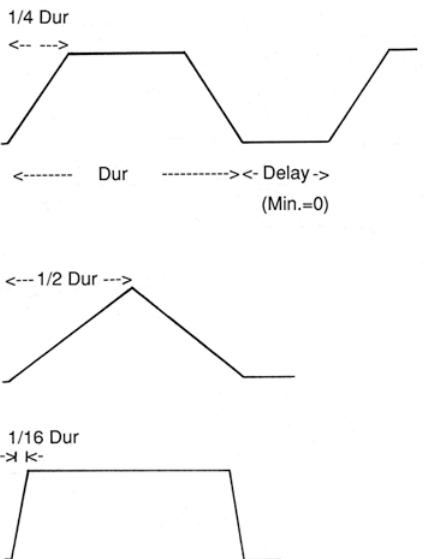
- A grain is a **small piece of sonic data**
- Usually have a duration  $\approx$  10 to 50 ms.
- The grain can be broken down into smaller components:

**Envelope** — used so no distortion and crunching noises at the beginning and end of the sample.

The shape of the envelope has a significant effect on the grain though.

- For a sampled sound, a short linear attack and decay prevent clicks being added to the sound.
- Changing the slope of the grain envelope Changes the resulting spectrum, Sharper attacks producing broader bandwidths, just as with very short grain durations.

**Contents** — The audio: derived from any source, basic waves or samples



GRAIN ENVELOPES



Back

Close

# Granular Synthesis: Making Sounds

Sounds made by the generation of thousands of short sonic grains:

- Combined linearly to form large scale audio events,
- 3 Possible combinations:

**Quasi-synchronous granular synthesis**

**Asynchronous granular synthesis**

**Pitch/Tempo-synchronous granular synthesis** — Preserve  
Pitch/Tempo whilst altering sample playback speed  
E.g. Intakt, Kontakt.

- The characteristics of the grains are definable which combine to give the characteristics of the overall sound.



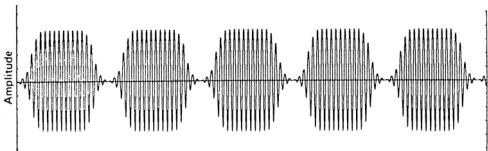
Back

Close

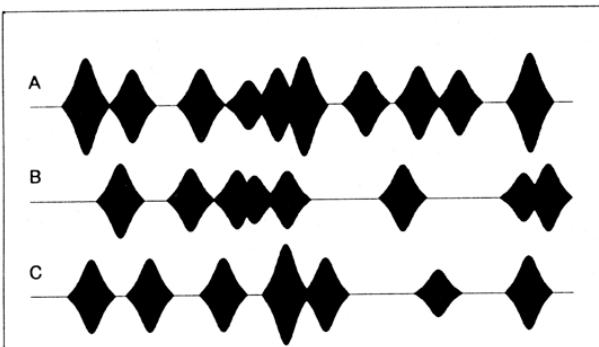
# Granular Synthesis: Making Sounds (Cont.)

## Quasi-synchronous granular synthesis:

- A grain stream of equal duration grains, produces Amplitude Modulation with grain durations less than 50 ms.

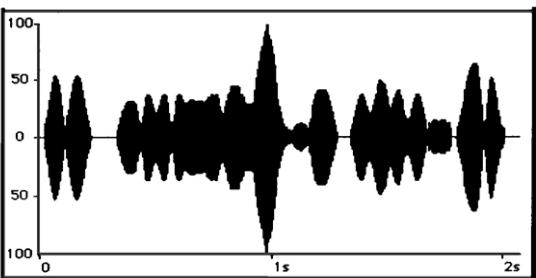


- Grain streams with variable delay time between grains — the sum of which resembles asynchronous granular synthesis.

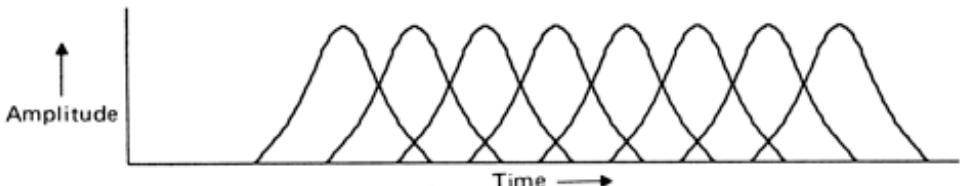


# Granular Synthesis: Making Sounds (Cont.)

**Asynchronous granular synthesis:** Grains are distributed stochastically with no quasi regularity.



**Pitch-synchronous granular synthesis:** Overlapping grain envelopes designed to be synchronous with the frequency of the grain waveform, thereby producing fewer audio artifacts.



# MATLAB Granular Synthesis Example

Simple MATLAB Example: granulation.m

```
% granulation.m
[filename,path] = uigetfile({'*.wav;*.waV;', 'Wav Files'; ...
    '.*', 'All files (*.*)'}, ...
    'Select a sound file');
if isequal(filename,0) | isequal(path,0)
cd(savedir);
return;
end
filenamepath = [path filename];
[x, fs] = wavread(filenamepath);

figure(1)
plot(x);

doit = input('\nPlay Original Wav file? Y/[N]:\n\n', 's');

if doit == 'y',
    playsound(x,fs);
end
```



# MATLAB Granular Synthesis Example (Cont.)

```
Ly=length(x);    y=zeros(Ly,1);                      %output signal
timex = Ly/fs;

% Constants
nEv=400; maxL=fs*0.02;   minL=fs*0.01;   Lw=fs*0.01;
% Initializations
L = round((maxL-minL)*rand(nEv,1))+minL;          %grain length
initIn = ceil((Ly-maxL)*rand(nEv,1));               %init grain
initOut= ceil((Ly-maxL)*rand(nEv,1));                %init out grain
a = rand(nEv,1);                                     %ampl. grain
endOut=initOut+L-1;
% Do Granular Synthesis
for k=1:nEv,
    grain=grainLn(x,initIn(k),L(k),Lw);
    figure(2)
    plot(grain);
    y(initOut(k):endOut(k))=y(initOut(k):endOut(k))+ grain;
end

figure(3)
plot(y)

doit = input('\nPlay Granular Synthesized Wave? Y/[N]:\n\n', 's');
if doit == 'y',
    playsound(y,fs);
end
```



Back

Close

# Physical Modelling

Physical modelling synthesis is the **synthesis of sound by using a mathematical model — set of equations and algorithms to simulate a physical source of sound.**

- Sound is then generated using parameters that describe the physical materials used in the instrument and the user's interaction with it,
- For example, by plucking/bowing a string, or covering toneholes on a flute, clarinet etc.
- For example, to model the sound of a drum, there would be a formula for how striking the drumhead injects energy into a two dimensional membrane.



Back

Close

# Physical Modelling: Examples

**Hardware :** Yamaha VL1 (1994), Roland COSM, Many since.

**Software :** Arturia Moog, Saxlab, Arturia Brass, PianoTeq

Examples of physical modelling algorithms:

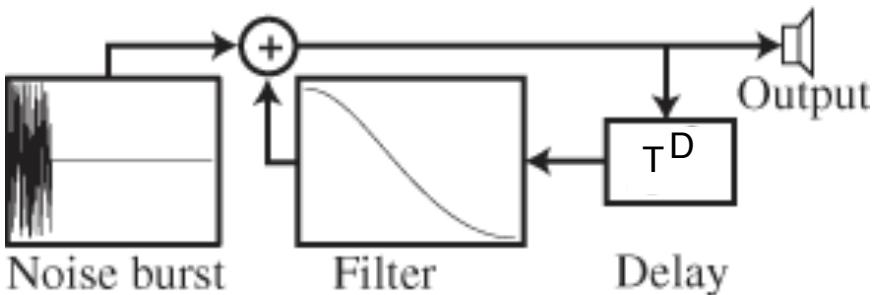
- Karplus-Strong synthesis (1971)
- Digital waveguide synthesis (1980s)
- Formant synthesis (1950s)



# Physical Modelling: Karplus-Strong Algorithm Basics

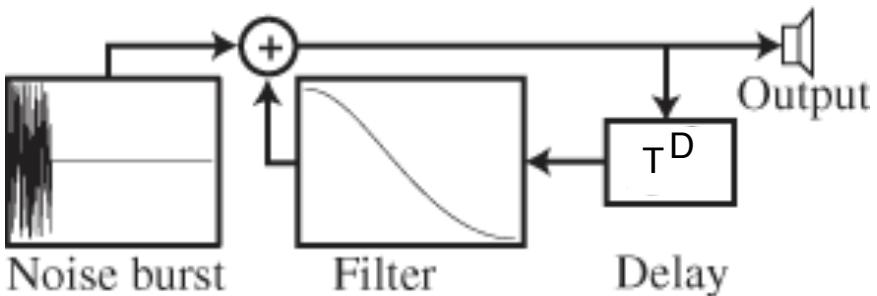
Simple Algorithm: *Makes a musical sound from noise*

- Loops a short noise burst through a filtered delay line to simulate the sound of a hammered or plucked string or some types of percussion.



- Feedback, Filtering and delay.
- Essentially subtractive synthesis technique based on a feedback loop similar to that of a comb filter.

# Physical Modelling: Karplus-Strong Algorithm More Details



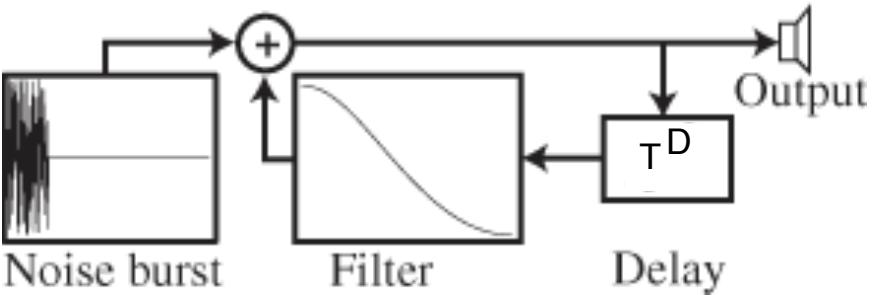
- Input: A burst of white noise,  $L$  samples long, (can use other signal).
- Output signal and feedback into a delay line.
- Output of the delay line is fed through a filter -gain of the filter must be less than 1 at all frequencies, usually a first order lowpass filter
- Filtered output is simultaneously mixed back into the output and fed back into the delay line.

# Physical Modelling: Karplus-Strong Algorithm Tuning

- Period of the resulting signal is the period of the delay line plus the average group delay of the filter;
- **Fundamental frequency** is the reciprocal of the period.
- Required delay  $D$  for a given fundamental frequency  $F_1$  is therefore calculated as:

$$D = \frac{F_s}{F_1}$$

where  $F_s$  is the sampling frequency.



# Physical Modelling: Karplus-Strong Algorithm MATLAB Example

MATLAB Karplus-Strong Algorithm: [karplus.m](#) ([Source](#))

```
% ***** Constants and Other Parameters *****
fs = 44100; % sampling rate
N = 80000; % length of vector to compute
D = 200; % delay line (or wavetable) length

% ***** Simple String Attenuation Filter *****
b = -0.99*[0.5 0.5];
z = 0;

% ***** Initialize delay lines *****
y = zeros(1,N); % initialize output vector
dline = 2 * rand(1, D) - 1.0;
ptr = 1;

figure(1); subplot(3,1,1);plot(dline);set(gca,'fontsize',18);
title('Original delayline');

subplot(3,1,2);plot(dline);set(gca,'fontsize',18);
title('Filter delayline step n');

loopsound(dline,fs,fs/D);

subplot(3,1,3); plot(y); title('Waveform Step n');set(gca,'fontsize',18);
```



Back

Close

```
figure(1);
% ***** Run Loop Start *****
for n = 1:N,
    y(n) = dline(ptr);
    [dline(ptr), z] = filter(b, 1, y(n), z);
    % Increment Pointers & Check Limits
    ptr = ptr + 1;
    if ptr > D
        ptr = 1;
    end
    if mod(n,2000) == 0
        subplot(3,1,2);plot(dline)
        str = sprintf('Filter delayline step %d',n);
        title(str);
        subplot(3,1,3); plot(y);
        str = sprintf('Waveform Step %d',n);
        title(str);
        figure(1);
    end
end

% Scale soundfile if necessary
max(abs(y))
if max(abs(y)) > 0.95
    y = y./ (max(abs(y))+0.1);
    disp('Scaled waveform');
end

figure(2);clf;plot(y); title('Final Step');set(gca,'fontsize',18);
playsound(y',fs);
```



Back

Close