CM3106 Chapter 5: Digital Audio Synthesis

Prof David Marshall
dave.marshall@cs.cardiff.ac.uk

and

Dr Kirill Sidorov
K.Sidorov@cs.cf.ac.uk
www.facebook.com/kirill.sidorov

School of Computer Science & Informatics
Cardiff University, UK

Digital Audio Synthesis

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**

Roadmap for Next Few Weeks of Lectures

Digital Audio Synthesis

We have talked a lot about synthesising sounds.

Several Approaches:

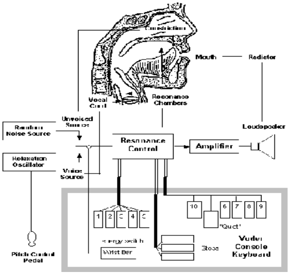
- Subtractive synthesis
- Additive synthesis
- FM (Frequency Modulation) Synthesis
- Sample-based synthesis
- Wavetable synthesis
- Granular Synthesis
- Physical Modelling

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).

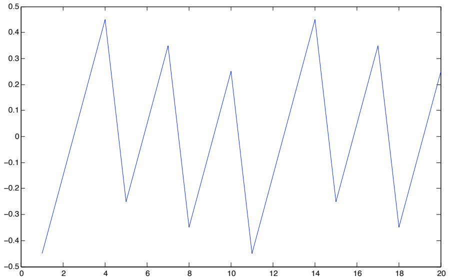
Popularised with Moog Synthesisers 1960-1970s

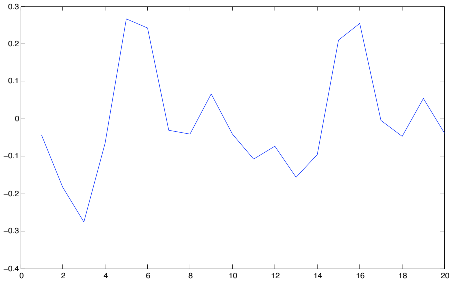


Subtractive synthesis: Simple Example

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.

Subtractive Synthesis: A Human Example

We can regard the way in which humans make noises as subtractive synthesis:

Oscillator — the vocal cords act as the sound source and

Filter — the mouth and throat modify the sound.

- 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.)

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.

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.

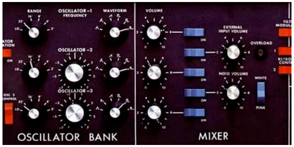
Subtractive synthesis: Electronic Control

Three Basic elements:

Source signal: Common source signals: square waves, pulse waves, sawtooth waves and triangle waves.

Modern synthesisers (digital and software) may include more complex waveforms or allow the upload of 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 (**strictly**: **not subtractive synthesis** but frequently used). Also used with **other** synthesis techniques.

Further Processing: ADSR Envelope

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

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.

Further Processing: ADSR Envelope (Cont.)

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

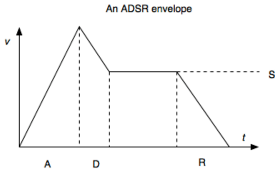
- For most mechanical instruments, this period is virtually instantaneous.
- For bowed strings or some popular synthesised "voices" that don't mimic real instruments, this parameter is slowed down. 'Slow attack' is commonly part of sounds — 'pads'.

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.



Using MATLAB Filter Example: Subtractive Synthesis Example

The example for studying subtractive synthesis uses the `butter()` and `filter()` MATLAB functions:

subtract_synth.m:

```
% 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));
sound(y,Fs);
end
```

Using MATLAB Filter Example: Subtractive Synthesis Example (cont)

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

Using MATLAB Filter Example: Subtractive Synthesis Example (cont)

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

%plot figures
doit = input('\Plot 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
```

synth.m

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

synth.m:

```
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
```

synth.m (cont)

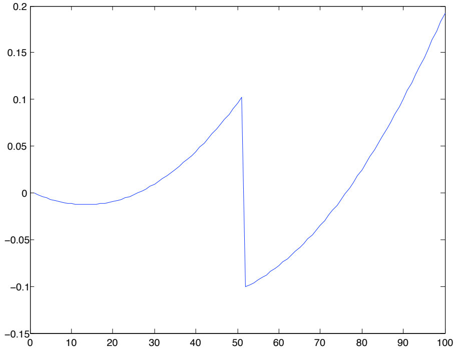
```
N = floor(dur*Fs);
n=0:N-1;
if (strcmp(type,'sine'))
  y = amp.*sin(2*pi*n*freq/Fs);

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));
```

synth.m (cont)

```
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
```

synth.m (Cont.)

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

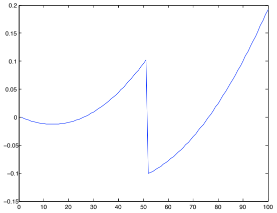


synth.m (Cont.)

This is created with (see synth.m):

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

Note: `fix()` rounds the elements of `X` to the nearest integers towards zero.



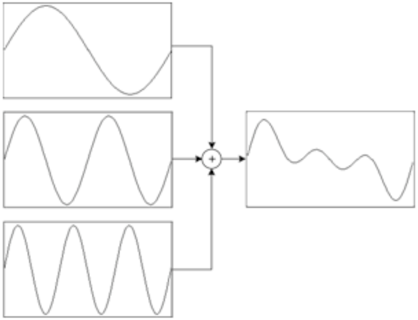
This form of sawtooth sounds slightly less harsh and is more suitable for audio synthesis purposes.

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

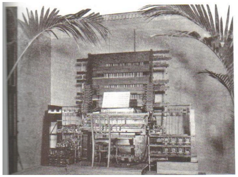
Organs: Pipe organs or Hammond organs. The concept of register-stops of organs = additive synthesis:

- 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)

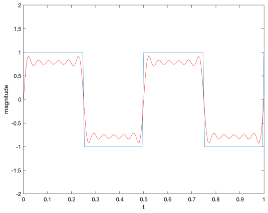


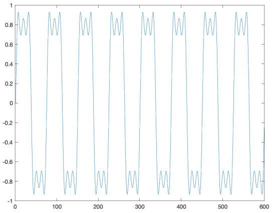
Additive synthesis: Basic Theory

- 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 in Ch5_2_Additive_Synthesis.mlx

Further Additive Synthesis Examples: Approximating a Square Wave

Back in the introduction to Fourier theory we showed how a Square wave is basically the addition of certain sine waves (and the Fourier transform can decompose the Square wave into sin waves).



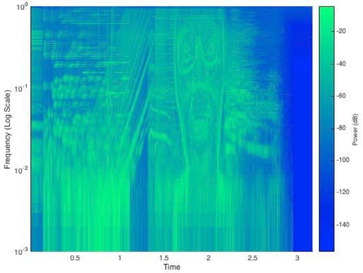


- We can grow our own in real space or Fourier space

For code example see: Ch5_2_Additive_Synthesis.mlx

Further Additive Synthesis Examples: Aphex Twin Demo

Recall recreating the Aphex Twin Spectrogram is another example of Inverse Fourier Transform Additive Synthesis where essentially "paints" in sinusoids based image intensity and pixel location and the adds them together.

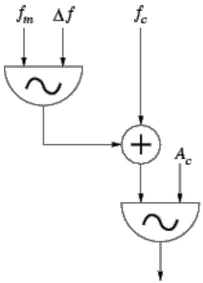


For code example see: Ch5_2_Additive_Synthesis.mlx

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.

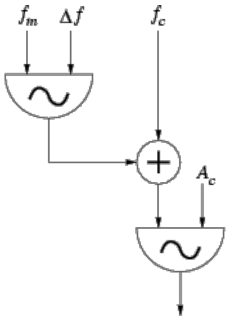
FM (Frequency Modulation) Synthesis (cont.)

- 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: Underpinnings

Definitions:

Oscillator:	A device for generating waveforms
Frequency 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:

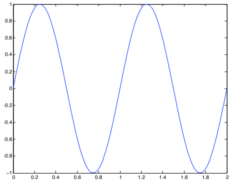
$$e = A\sin(\alpha t + I\sin\beta t)$$

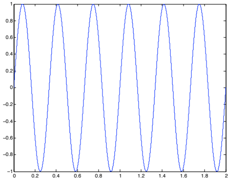
A is the peak amplitude

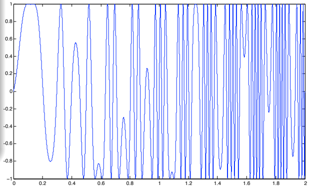
e is the instantaneous amplitude of the modulated carrier

α and β are the respective carrier and modulator frequencies

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







FM MATLAB Example

MATLAB code to produce basic FM (fm_eg.m), see also fm_eg_plot.m:

fm_eg.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));

sound(y, fs);
```

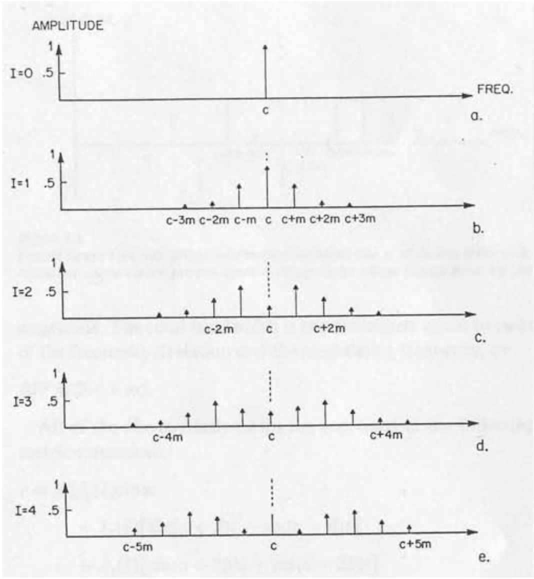
FM Synthesis: Side Frequencies

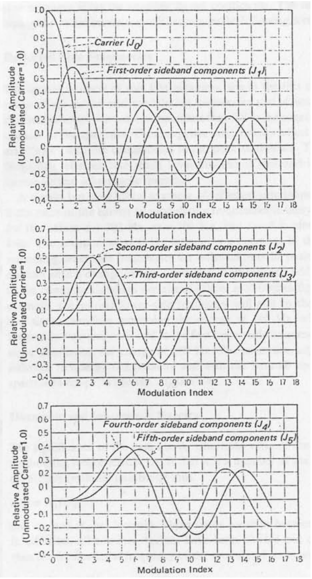
The **harmonic distribution** of a simple sine wave signal modulated by another sine wave signal can be represented with **Bessel functions**:

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

- 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.

FM Synthesis: Side Frequencies (Cont.)





A few insights as to how Bessel functions

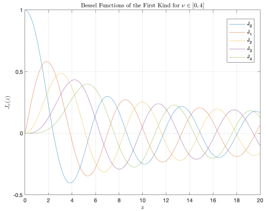
A few insights as to how Bessel functions can help explain why FM synthesis sounds the way it does:

- $J_0(I)$ decides the amplitude of the carrier.
- $J_1(I)$ controls the first upper and lower sidebands.
- Generally, $J_n(I)$ governs the amplitudes of the nth upper and lower sidebands.
- Higher-order Bessel functions start from zero more and more gradually, so higher-order sidebands only have significant energy when I is large.
- The spectral bandwidth increases with I; the upper and lower sidebands grow toward higher and lower frequencies, respectively.
- As I increases, the energy of the sidebands vary much like a damped sinusoid.

MATLAB knows about Bessel functions:

Ch5_3_FM_Synthesis.mlx

```
z = 0:0.1:20;
J = zeros(5,201);
for i = 0:4
    J(i+1,:) = besselj(i,z);
end

plot(z,J)
grid on
legend('J_0','J_1','J_2','J_3','J_4','Location','Best')
title('Bessel Functions of the First Kind for $\nu \in [0, 4]$',
        'interpreter','latex')
xlabel('z','interpreter','latex')
ylabel('$J_\nu(z)$','interpreter','latex')
```



See also: Ch5_3_FM_Synthesis.mlx for further details

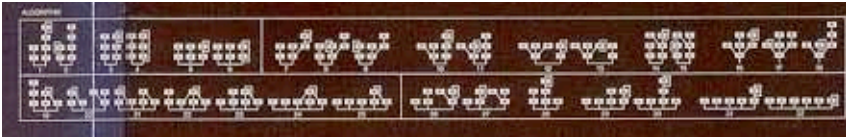
FM Synthesis: Making Complex Sounds

Operators and Algorithms

Operators are just Oscillators in FM Terminology.

- FM synths will have either 4 or 6 Operators.
- **Why so many Operators?**
  Sounds from one Modulator and one Carrier aren't exactly that overwhelmingly complex

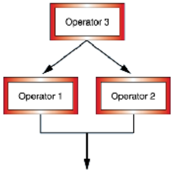
Algorithms are the preset combinations of routing available to you.

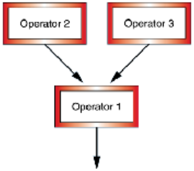


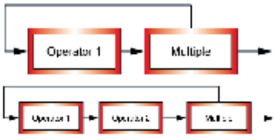
FM Synthesis: FM Algorithms

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







Some More FM Waveform Examples

See Ch5_3_FM_Synthesis.mlx for some further practical examples of how to synthesise:

- A sine wave which "compresses" and "uncompress" in time
- A sine wave which undergoes an periodic modulation
- A Bell Sound
- A wood block strike type sound
- Brass sounds

Sample-based synthesis

Basic Ideas: Similar to 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.

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.

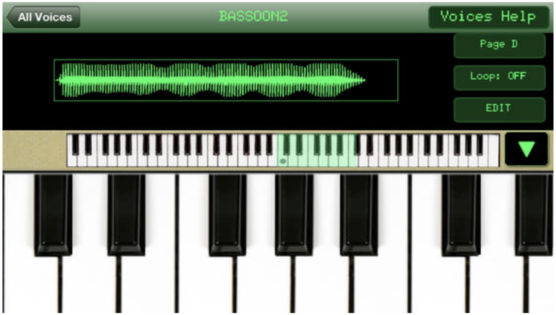
Sample-based synthesis: Examples

- 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

Sampling Trivia

CMI Fairlight

- Cost the price of a good house (c. £20,000) when released in 1979.
- It is now available as an iPad App!
  - Fully functional



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 of Sampling (c. Late 1980s/Early 90s)

Computer memory expensive:

- Samples had to be as **short** and as **few** as possible.

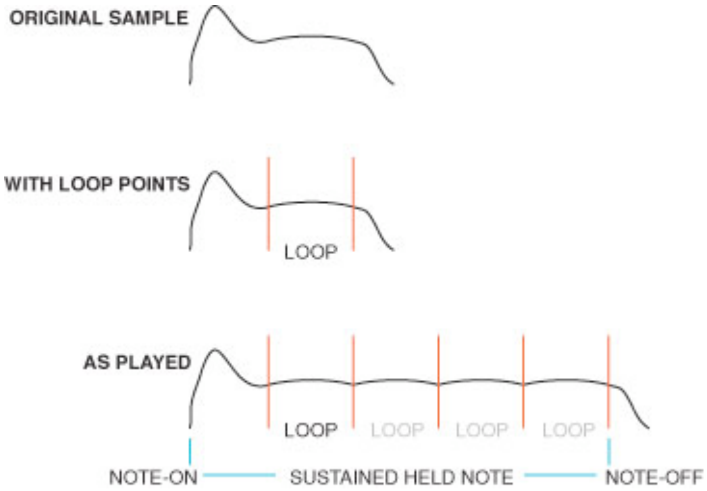
This was achieved by **looping a part of the sample**

Looping today:

Looping still useful for

- Saving sample memory space — efficiency
- Looping audio material: Drum tracks, sound effects, *etc.*

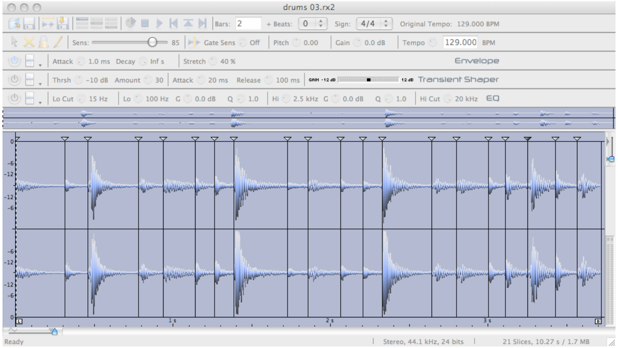
Sample-based synthesis Basics: Looping (Cont.)



Problem: How to find looping points?

Finding looping points

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



drums 03.rx2

Bars: 2 + Beats: 0 Sign: 4/4 Original Tempo: 129.000 BPM

Sens: 85 Gate Sens Off Pitch 0.00 Gain 0.0 dB Tempo 129.000 BPM

Attack 1.0 ms Decay Inf s Stretch 40 %

Thrsh -10 dB Amount 30 Attack 20 ms Release 100 ms GAIN -12 dB 12 dB

Lo Cut 15 Hz Lo 100 Hz G 0.0 dB Q 1.0 Hi 2.5 kHz G 0.0 dB Q 1.0 Hi Cut 20 kHz

Ready | Stereo, 44.1 kHz, 24 bits | 21 Slices, 10.27 s / 1.7 MB

  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.

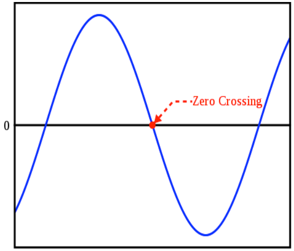
Finishing off the loop:

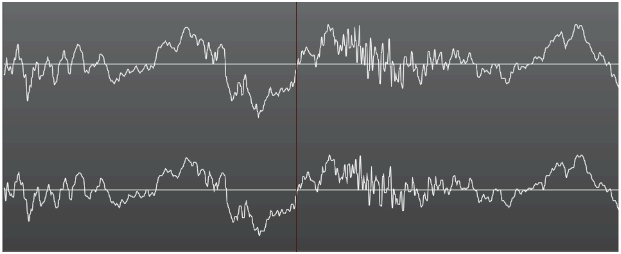
- Early Days: Use a volume envelope curve to make the sound fade away.
- Today: Include tail off sample in data — triggered by note off.

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 for many practical looping applications - **how to detect correct loop point**?





Beat Perception:

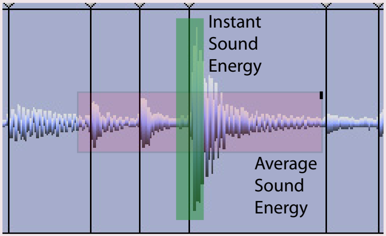
The human listening system determines the rhythm of music by detecting a pseudo — periodical succession of beats

- 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 sound's 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,

Example of using Human Perception — a theme of this module.

Beat Slicing Algorithms Ideas (1):

Simple sound energy beat detection:



- 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.

Beat Slicing Algorithms Ideas (2):

Frequency selected sound energy:

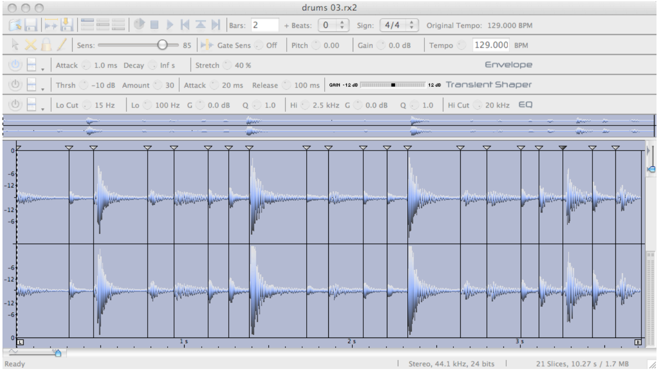
More elaborate model:

Try to detect large 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 section.

Beat Slicing in Recycle — Slice Creation

- Launch ReCycle and select and open a file.
- In the main window, you may click the Play button to hear the entire loop, from start to end (repeating until you click the Stop button.)
- **To create slices**: Adjust the Sensitivity slider to the right — the exact value depends on the audio.


drums 03.rx2

Bars: 2 + Beats: 0 Sign: 4/4 Original Tempo: 129.000 BPM

Sens: 85 Gate Sens Off Pitch 0.00 Gain 0.0 dB Tempo 129.000 BPM

Attack 1.0 ms Decay Inf s Stretch 40 % Envelope

Thrsh -10 dB Amount 30 Attack 20 ms Release 100 ms GAIN -12 dB 12 dB Transient Shaper

Lo Cut 15 Hz Lo 100 Hz G 0.0 dB Q 1.0 Hi 2.5 kHz G 0.0 dB Q 1.0 Hi Cut 20 kHz EQ

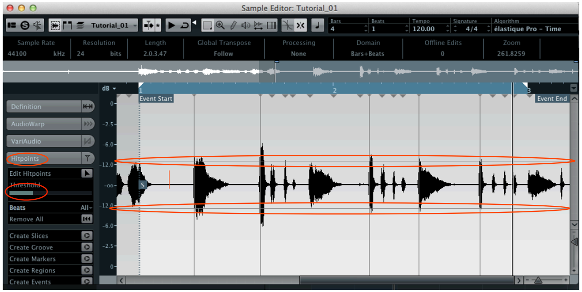
Ready | Stereo, 44.1 kHz, 24 bits | 21 Slices, 10.27 s / 1.7 MB

Beat Slicing in Cubase

Cubase takes a simpler more user interactive approach to beat slicing.

In the **Sample Editor**

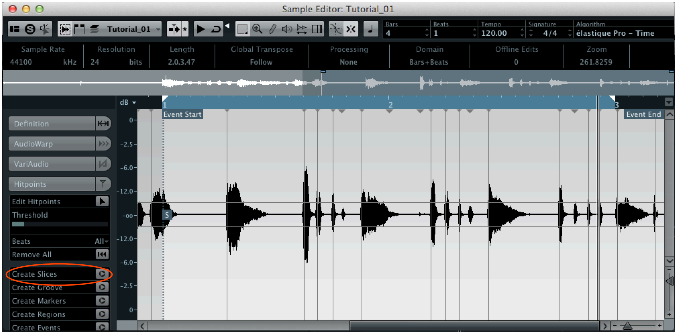
- Select **Hitpoints** Editing Option.
- Either adjust **Threshold** or visual **horizontal lines** to select the appropriate level of hit points, as displayed.


Sample Editor: Tutorial_01

Bars 4 | Beats 1 | Tempo 120.00 | Signature 4/4 | Algorithm élastique Pro – Time

Sample Rate 44100 kHz | Resolution 24 bits | Length 2.0.3.47 | Global Transpose Follow | Processing None | Domain Bars+Beats | Offline Edits 0 | Zoom 261.8259

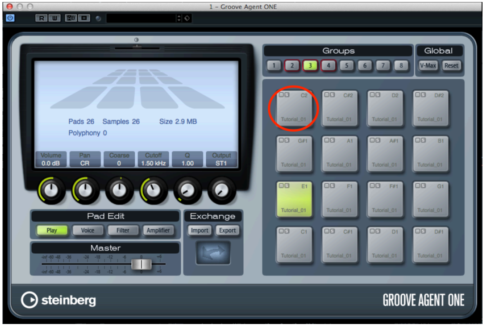
Beat Slicing in Cubase (Cont.)



- When happy hit the **Create Slices** button.
- Hitpoints can then be edited in a similar fashion to Recycle

Cubase Drum Sampler

Cubase has a built in Drum Sampler: **Groove Agent One**.



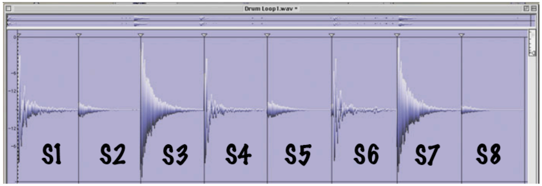
To map sliced beats (hit points) to Groove Agent One:

- Simply drag the sliced audio file onto one of the **Drum Pads**.
  - Subsequent slices are mapped to consecutive pads.

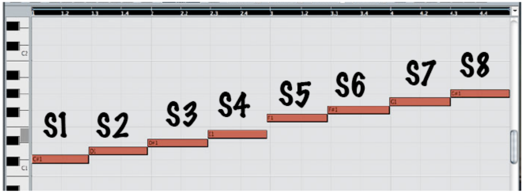
Beat Slicing — Simple Application

Recap: Create a MIDI performance of a chromatic scale, whose note timing trigger each sample at the perfect time to recreate the original audio.

Recycle Slicing:

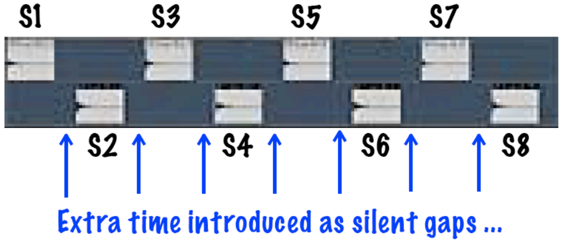


Midi Mapping/Triggering of Slices:

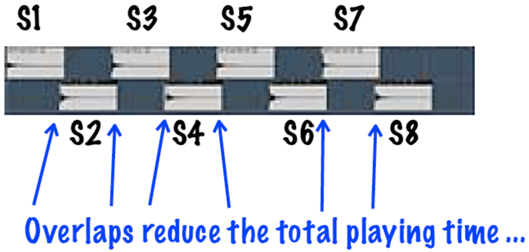


Beat Slicing — Tempo Change Problems

Replay after tempo is made slower:



Replay after tempo is made faster:

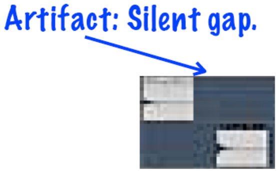


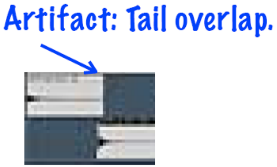
Beat Slicing - Artefacts, Solving the Tempo Problem

For drum loops etc. — **attacks** are artefact free.

- The most important part of a percussion sound.

Two artefacts (from previous slide):





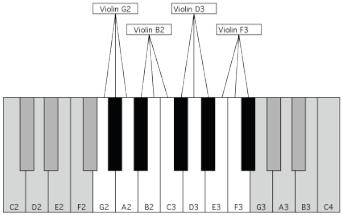
Beat Slicing Solution

Solutions

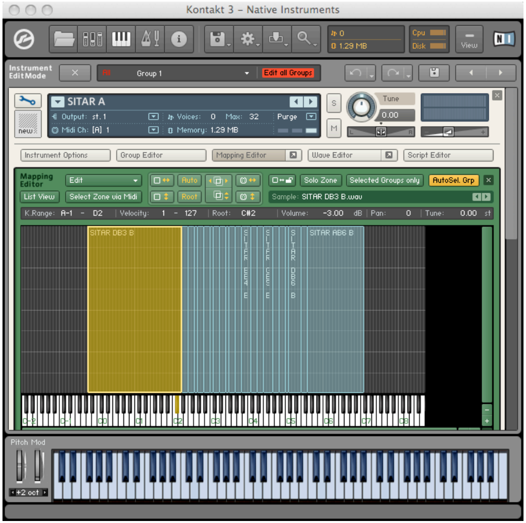
- Apply envelope to each slice to fade it to silence before gap or overlap.
- **For gaps**: loop the end of the tail to extend it through the gap.

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


Kontakt 3 – Native Instruments

SITAR A

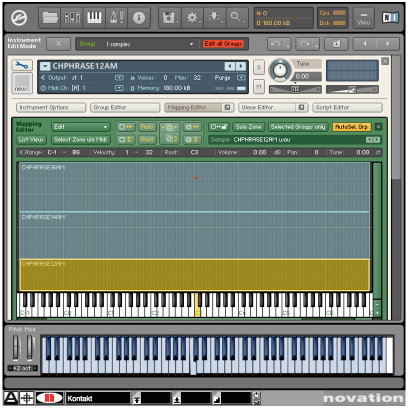
Output: st. 1
Voices: 0 Max: 32
Midi Ch: [A] 1
Memory: 1.29 MB
Tune 0.00

Sample: SITAR DB3 B.wav

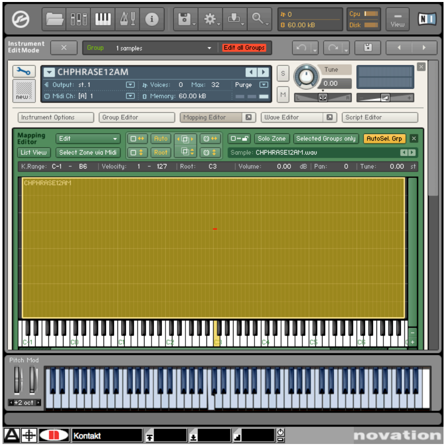
K.Range: A-1 - D2 | Velocity: 1 - 127 | Root: C#2 | Volume: -3.00 dB | Pan: 0 | Tune: 0.00 st

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**


Group 1 samples
CHPHRASE12AM
Output: st. 1 Voices: 0 Max: 32
Midi Ch: [A] 1 Memory: 180.00 kB
Tune 0.00
Sample: CHPHRASE12AM.wav
K.Range: C-1 - B6 | Velocity: 1 - 32 | Root: C3 | Volume: 0.00 dB | Pan: 0 | Tune: 0.00 st

Sample-based Synthesis: Velocity Layers (1)



Output: st. 1 | Voices: 0 Max: 32 | Midi Ch: [A] 1 | Memory: 60.00 kB

Sample: CHPHRASE12AM.wav

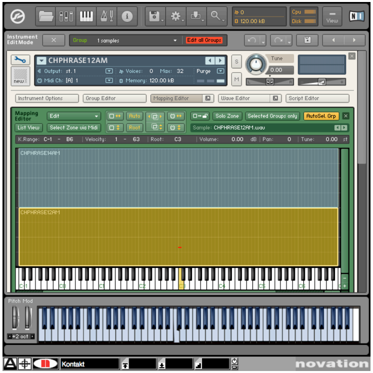
K.Range: C-1 - B6 | Velocity: 1 - 127 | Root: C3 | Volume: 0.00 dB | Pan: 0 | Tune: 0.00 st

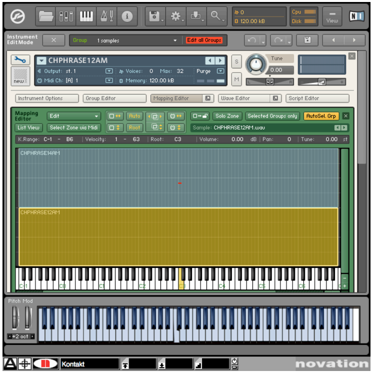
(Single key mapped) Single Velocity Layer — Only **one** type of sound played at **any velocity**.

Volume output maybe controlled by velocity but **no change** in **timbre** of sound.

Sample-based Synthesis: Velocity Layers (2)

(Single key mapped) Dual Velocity Layer:

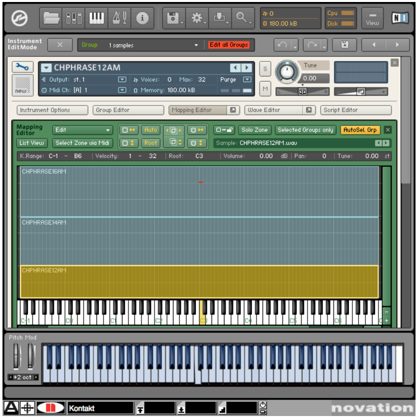

CHPHRASE12AM
Output: st. 1 Voices: 0 Max: 32 Midi Ch: [A] 1 Memory: 120.00 kB
Sample: CHPHRASE12AM.wav
K.Range: C-1 - B6 Velocity: 1 - 63 Root: C3 Volume: 0.00 dB Pan: 0 Tune: 0.00 st


CHPHRASE12AM
Output: st. 1 Voices: 0 Max: 32 Midi Ch: [A] 1 Memory: 120.00 kB
Sample: CHPHRASE12AM.wav
K.Range: C-1 - B6 Velocity: 1 - 63 Root: C3 Volume: 0.00 dB Pan: 0 Tune: 0.00 st

Sound **one** played at **lower level** velocity

Sound **two** played at **higher velocity**

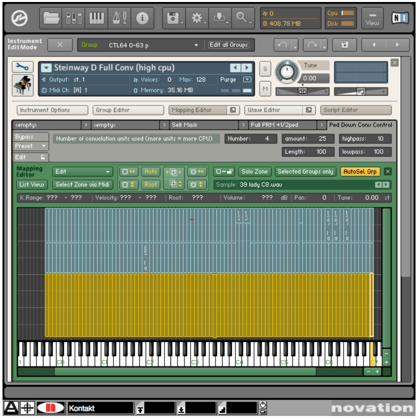
Sample-based Synthesis: Velocity Layers (3)



(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


Steinway D Full Conv (high cpu)
Output: st. 1 Voices: 0 Max: 128 Purge
Midi Ch: [A] 1 Memory: 35.16 MB
Tune 0.00
Group CTL64 0-63 p Edit all Groups
Number of convolution units used (more units = more CPU) Number: 4 amount: 25 highpass: 10
Length: 100 lowpass: 100
Sample: 39 lady C8.wav
K.Range: ??? - ??? | Velocity: ??? - ??? | Root: ??? | Volume: ??? dB | Pan: 0 | Tune: 0.00 st

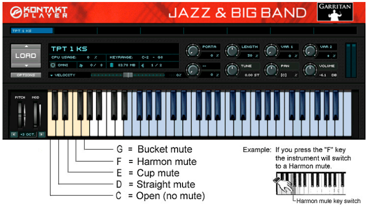
Most instruments are a combination of multisample key mapped and velocity layers

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**:

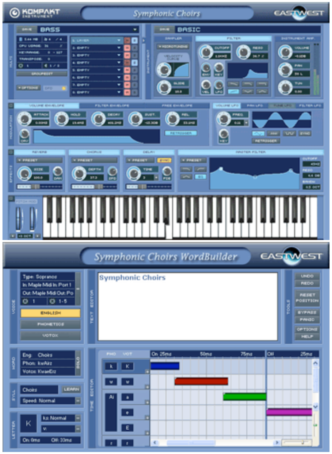
  Use keys (usually lower keys outside of instrument range) to select appropriate sounds

  - Essentially banks of key mapped velocity layered samples



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).



A Symphonic Choir Sample Library

Source: video

Sample Based Synthesis Further References

- *www.cs.berkeley.edu/ lazzaro/class/music209* — Good overview of Beat slicing. (I borrowed a few figures from here)
- Sound on Sound Magazine Beat Slicing Masterclass — *www.soundonsound.com/sos/jun04/articles/beatslicing.htm*
- *emusician.com/mag/square_one/emusic_slice/index.html* — Electronic Musician Magazine Article on Beat Slicing

Wavetable synthesis

What is wavetable 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.

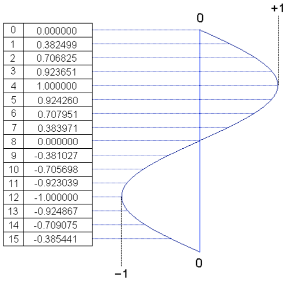
Not to be confused with common PCM sample buffer playback: soundcards

Wavetable synthesis: Examples

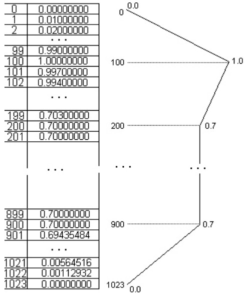
PPG Wave Series:	Implementation of wavetable synthesis employed an array containing 64 pointers to individual single-cycle waves.
Waldorf Microwave:	Next generation PPG.
Roland D-50	(and Roland MT-32/variants:) "Linear Arithmetic" synthesizers — combined complex sampled attack phases with less complex sustain/decay phases (basically a wavetable synthesizer with a 2-entry wave sequence table).
Prophet-VS,	(Sequential Circuits)
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



0	0.000000
1	0.382499
2	0.706825
3	0.923651
4	1.000000
5	0.924260
6	0.707951
7	0.383971
8	0.000000
9	-0.381027
10	-0.705698
11	-0.923039
12	-1.000000
13	-0.924867
14	-0.709075
15	-0.385441



0	0.00000000
1	0.01000000
2	0.02000000
	. . .
99	0.99000000
100	1.00000000
101	0.99700000
102	0.99400000
	. . .
199	0.70300000
200	0.70000000
201	0.70000000
	. . .
899	0.70000000
900	0.70000000
901	0.69435484
	. . .
1021	0.00564516
1022	0.00112932
1023	0.00000000

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).

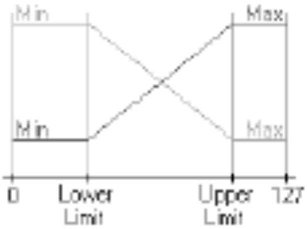
Wavetable v. Sample Playback

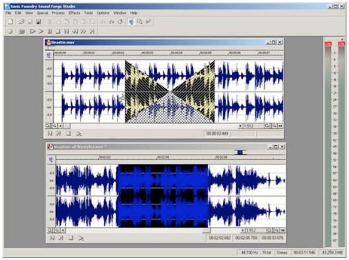
- **Differs** from simple sample playback as
  - Output waveform is always generated in real time as the CPU processes the wave sequences
  - Waves in the tables are rarely more than 1 or 2 periods in length.

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 Example

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

wavetable_synth.m:

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

%Create a single sine waves
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 wave
y2 = synth(f2,1/f2,0.9,Fs,'saw');

doit = input('\nPlay/Plot Raw saw
   y2 looped for 10 ...
   seconds? Y/[N:]\n\n', 's');
if doit == 'y',
figure(2)
plot(y2);
loopsound(y2,Fs,10*Fs/f2);
end
```

wavetable_synth.m (Cont.)

Making the crossfades

```
%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;
```

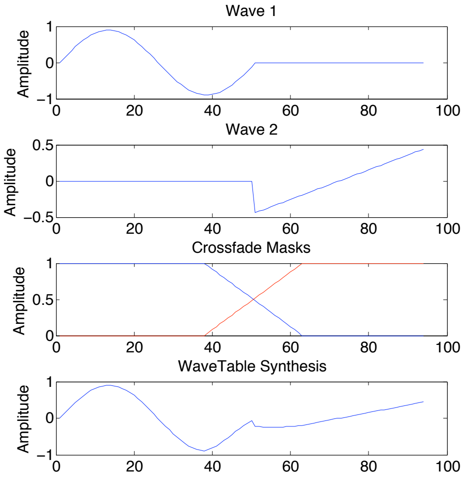
```
% Apply crossfade centered over
% the join of y1 and y2
pad = (Fs/f1) + (Fs/f2)
          - 2.5*xfadewidth;
xramp1 = [ones(1,1.5*xfadewidth),
            ramp2, zeros(1,floor(pad))]
xramp2 = [zeros(1,1.5*xfadewidth),
             ramp1, ones(1,floor(pad))]

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

ywave = ywave2;
```

wavetable_synth.m (Cont.)

Adding the crossfade

```
% do xfade

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);
 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
```

MATLAB Example: Linear Crossfading (Cont.)

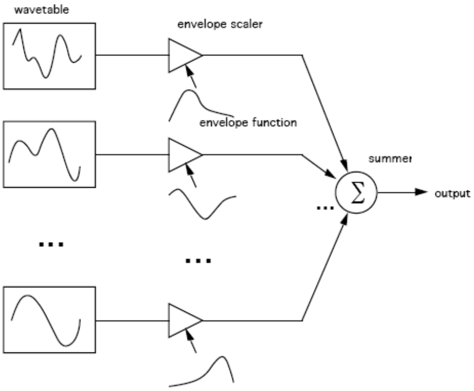


Note: This sort of technique is useful to create an ADSR envelope in MATLAB

Wavetable synthesis: Dynamic Waveshaping (2)

More sophisticated method: Sequential Enveloping

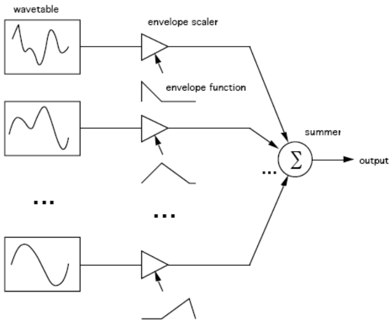
- Example below: two wavetables are being mixed at any one instance of time by moving envelope scale



Wavetable synthesis: Dynamic Waveshaping (Cont.)

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 synthesising 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.

Wavetable synthesis: Advantages (cont.)

Enabling Faster Playback

- 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.

Built in MATLAB Wavetable examples

MATLAB has a basic wavetable synthesiser built-in to its Audio Toolbox:

```
doc wavetableSynthesizer
```

See this MATLAB page for some examples

- Generate Variable-Frequency Staircase Wave:
- Manipulate Audio Samples Using Wavetable Synthesizer
- Modify Wavetable While Stream Processing
- Tune Wavetable Synthesizer Parameters

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.

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 musical?

- Usable as music or soundscapes (ambient)
- Usable as Sound effects
- **MUSICAL:** 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.

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!

Granular Synthesis: Implementations

- Software:

  Many implementations nowadays:

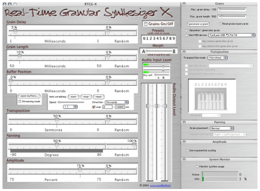
  Programmable: Csound, MATLAB, MAX/MSP routines:

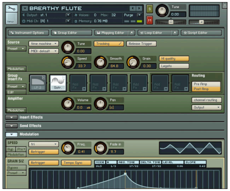
  Standalone: Supercollider, Granulab, RTGS X.

  DAW plug-ins standalone: VSTis etc.

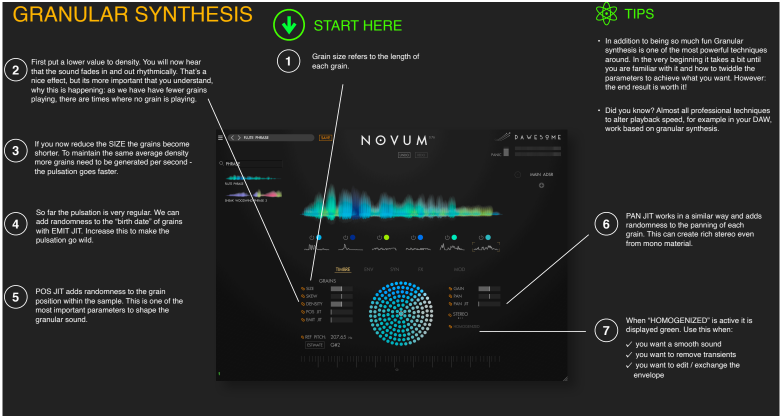
  Modern Music Samplers: Native Instruments' Kontakt, Intakt...., others

- Hardware: Korg Kaos Pad.





Novum Granular Synthesis



GRANULAR SYNTHESIS

START HERE

1 Grain size refers to the length of each grain.

2 First put a lower value to density. You will now hear that the sound fades in and out rhythmically. That's a nice effect, but its more important that you understand, why this is happening: as we have have fewer grains playing, there are times where no grain is playing.

3 If you now reduce the SIZE the grains become shorter. To maintain the same average density more grains need to be generated per second - the pulsation goes faster.

4 So far the pulsation is very regular. We can add randomness to the "birth date" of grains with EMIT JIT. Increase this to make the pulsation go wild.

5 POS JIT adds randomness to the grain position within the sample. This is one of the most important parameters to shape the granular sound.

6 PAN JIT works in a similar way and adds randomness to the panning of each grain. This can create rich stereo even from mono material.

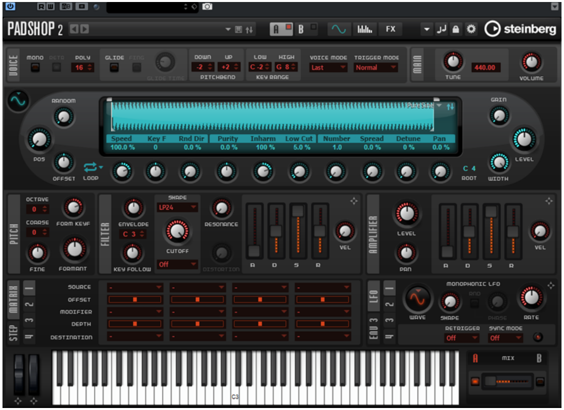
7 When "HOMOGENIZED" is active it is displayed green. Use this when:

- ✓ you want a smooth sound
- ✓ you want to remove transients
- ✓ you want to edit / exchange the envelope

TIPS

- In addition to being so much fun Granular synthesis is one of the most powerful techniques around. In the very beginning it takes a bit until you are familiar with it and how to twiddle the parameters to achieve what you want. However: the end result is worth it!
- Did you know? Almost all professional techniques to alter playback speed, for example in your DAW, work based on granular synthesis.

Cubase Padshop Granular Synthesiser



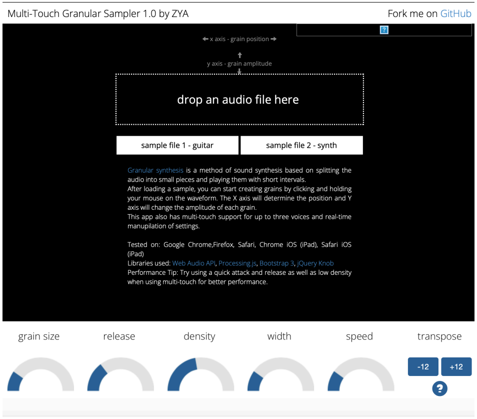
Poly 16; Pitchbend Down -2, Up +2; Key Range Low C -2, High G 8; Voice Mode Last; Trigger Mode Normal; Tune 440.00

Speed	Key F	Rnd Dir	Purity	Inharm	Low Cut	Number	Spread	Detune	Pan
100.0 %	0	0.0 %	0.0 %	100 %	5.0 %	1.0	0.0 %	0 %	0.0 %

Root C 4; Octave 0; Coarse 0; Shape LP24; Key Follow C 3; Distortion Off; Retrigger Off; Sync Mode Off

Web Audio Granular Synthesisers

- https://zya.github.io/granular/


Multi-Touch Granular Sampler 1.0 by ZYA

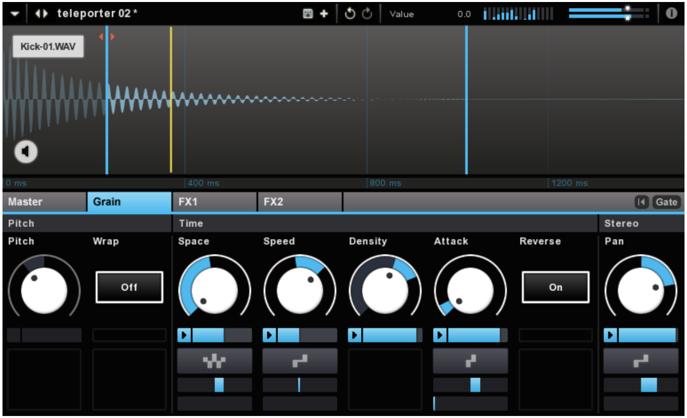
Fork me on GitHub

Granular synthesis is a method of sound synthesis based on splitting the audio into small pieces and playing them with short intervals.
After loading a sample, you can start creating grains by clicking and holding your mouse on the waveform. The X axis will determine the position and Y axis will change the amplitude of each grain.
This app also has multi-touch support for up to three voices and real-time manupilation of settings.

Tested on: Google Chrome,Firefox, Safari, Chrome iOS (iPad), Safari iOS (iPad)
Libraries used: Web Audio API, Processing.js, Bootstrap 3, jQuery Knob
Performance Tip: Try using a quick attack and release as well as low density when using multi-touch for better performance.

Granite Granular Syntehiser

http://www.newsonicarts.com/html/granite.php



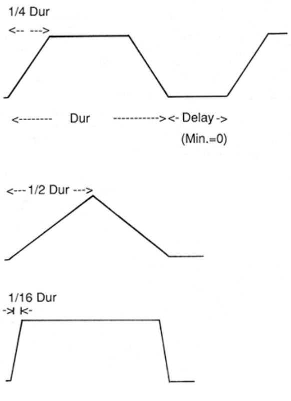
Note: Commercial application but demo available)

Plenty of other example — just search

Granular Synthesis: What is a grain?

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



GRAIN ENVELOPES

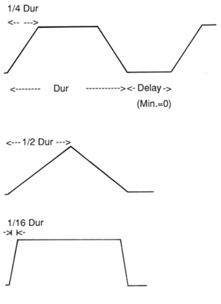
Granular Synthesis: What is a grain?

Grain 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 sound.

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

Contents: The audio: derived from any source: **basic waveforms** or **samples**



GRAIN ENVELOPES

Granular Synthesis: Making Sounds

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

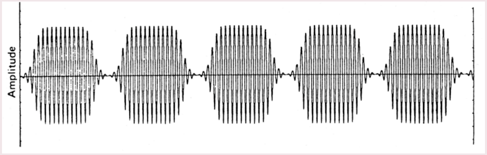
- Combined linearly to form large scale audio events,
- 3 Possible combinations:

  Quasi-synchronous granular synthesis
  Asynchronous granular synthesis
  Pitch/Tempo-synchronous granular synthesis
- The grains' characteristics are also definable and when combined affect the overall sound.

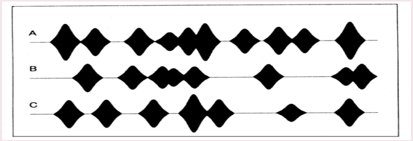
Granular Synthesis: Making Sounds (Cont.)

Quasi-synchronous granular synthesis:

- A grain stream of equal duration grains, produces amplitude modulation (**see later**) with grain durations **less** than 50 ms.



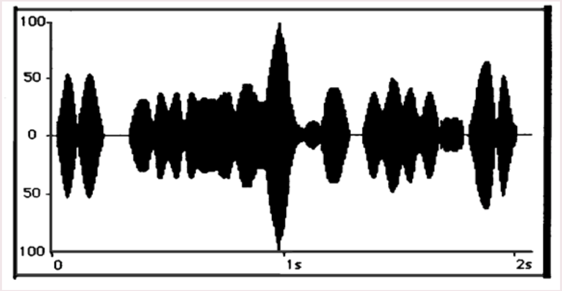
- Grain streams with variable delay time between grains: the sum of which resembles asynchronous granular synthesis.



Granular Synthesis: Making Sounds (Cont.)

Asynchronous granular synthesis:

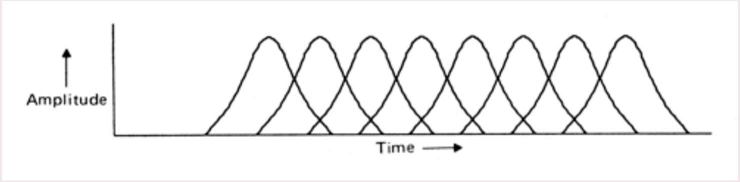
Grains are distributed stochastically with no quasi regularity.



Granular Synthesis: Making Sounds (Cont.)

Pitch/Tempo-synchronous granular synthesis:

- Preserve Pitch/Tempo whilst altering sample playback speed E.g. Intakt, Kontakt.
- Overlapping grain envelopes designed to be **synchronous** with the **frequency** of the grain **waveform**, thereby producing fewer audio artifacts.



Granular Synthesis MATLAB Example

Simple MATLAB Example: 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] = audioread(filenamepath);

figure(1)
plot(x);

doit = input('\nPlay Original Wav file? Y/[N:]\n\n', 's');
if doit == 'y',
 sound(x,fs);
end
```

MATLAB Granular Synthesis Example (Cont.)

granulation.m (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

% Plot figure and play sound
 .......
```

MATLAB Granular Synthesis Example (Cont.)

grainLn.m

```
function y = grainLn(x,iniz,L,Lw)
% extract a long grain
% x     input signal
% init  first sample
% L     grain length (in samples)
% Lw    length fade-in and fade-out (in samples)

if length(x) <= iniz+L , error('length(x) too short.'),  end

y = x(iniz:iniz+L-1);                              % extract segment
w = hanning(2*Lw+1);
y(1:Lw)       = y(1:Lw).*w(1:Lw);                  % fade-in
y(L-Lw+1:L) = y(L-Lw+1:L).*w(Lw+2:2*Lw+1); % fade-out
```

More Advanced Granular Synthesis

Above is quite simple and general and can be employed to obtain very different sounds and sound effects.

More control over the sound:

- The above sonds is greatly influenced by the criterion used to choose the instants .
- If these points are regularly spaced in time and the grain waveform does not change too much,
    - the technique can be interpreted as a **filtered pulse train**, i.e. it produces a periodic sound whose spectral envelope is determined by the grain waveform interpreted as impulse response.

PSOLA based Pitch/Tempo-synchronous granular synthesis

The above is an example is the **PSOLA based Pitch/Tempo-synchronous granular** synthesis (**more soon**), where:

- When the distance between two subsequent grains is much greater than L_k, the sound will result in grains separated by interruptions or silences with a specific character.
- When many short grains overlap (i.e. the distance is less than L_k), a sound texture effect is obtained.

See accompanying lab exercise

Short and Overlapping Grains

Short Grains

- The above code, for simplicity of illustration, only uses long grains.
- experiment by mixing or swapping in short grains via the `grainSh.m` function — **See accompanying lab exercise**

Overlapping Grains

It is quite simple to extend the code above to account for overlapping grains:

- To overlap a grain g_k at instant $n_k =$ `iniOLA` with amplitude a_k, **See accompanying lab exercise**.

```
endOLA = iniOLA+length(grain)-1;
y(iniOLA:endOLA) = y(iniOLA:endOLA) + ak * grain;
```

PSOLA based Pitch/Tempo-synchronous granular synthesis

PSOLA exists as common means of pitch and tempo shifting outside of any synthesis method.

- Historically, predates the phase vocoder but still common approach.
- Historically important to the development of Granular synthesis.
- PSOLA originated for speech processing, paarticularly speech synthesis,
    - It also applicable to musical applications.

PSOLA in action

Not unlike the phase vocoder:

- Used to modify the pitch (scaling) and duration (time stretching) of a speech signal.
- PSOLA works by dividing the speech waveform in small overlapping segments.
  - To change the pitch of the signal, the segments are moved further apart (to decrease the pitch) or closer together (to increase the pitch).
  - To change the duration of the signal, the segments are then repeated multiple times (to increase the duration) or some are eliminated (to decrease the duration).
  - The segments are then combined using the overlap add technique.
- **The difference between PSOLA and the phase vocoder is there is no STFT in PSOLA.**

See Live Scripts for more details and code examples:
Ch5_6_Granular_Synthesis.mlx

Physical Modelling

Physical modelling synthesis

The synthesis of sound by using a mathematical model: sets of equations and algorithms to simulate a physical source of sound.

- Sound is generated using model 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.

Physical Modelling: Examples

Hardware:	Yamaha VL1 (1994), Roland COSM, Many since.
Software:	Arturia Moog, PianoTeq

Examples of physical modelling algorithms:

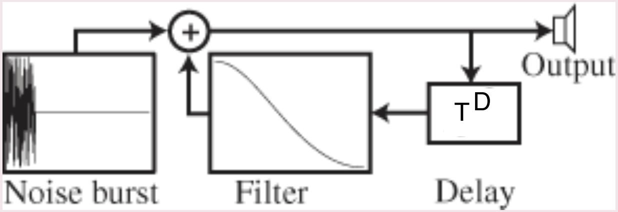
- Karplus-Strong strong synthesis (1971)
- Digital waveguide synthesis (1980s)
- Formant synthesis (1950s)

Physical Modelling Exampe

Karplus-Strong Algorithm

Simple Algorithm: *Makes a musical sound from noise*

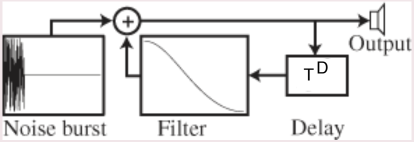
- Loops a short noise burst through a filtered delay line to simulate the sound of a hammered or plucked string or some types of percussion.



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

Physical Modelling Example

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.

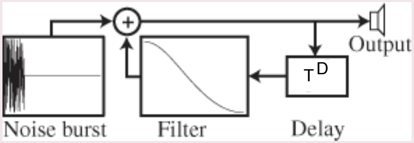
Physical Modelling Example

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 MATLAB Example

MATLAB Karplus-Strong Algorithm: karplus.m:

```
% ******* 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');
```

Physical Modelling MATLAB Example (Cont.)

karplus.m

```
loopsound(dline,fs,fs/D);

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

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
```

Physical Modelling MATLAB Example (Cont.)

karplus.m

```
  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);
sound(y',fs);
```

See also Ch5_7_Physical_Modelling_Synthesis.mlx for results

Physical Modelling MATLAB Example: Drum Sound

The basic algorithm is as follows:

- Start with wavetable X, of length p,
    - such that
      $$X(t) = +1/2(X(t-p) + X(t-p+1))$$
      with probability b, and
    - $$X(t) = -1/2(X(t-p) + X(t-p+1))$$
      with probability $1-b$ for $t > p$.
- Since b introduces randomness into the sound, the initial wavetable can be anything from a completely random signal to a sine wave to a constant.
- The wavetable length p affects the decay rate of the sound (big = long decay) as well as the pitch somewhat (big = low pitch).
    - p should be in a range from about $150 - -500$.
- The probability b is called the **blend factor** and can range from 0 to 1.
    - $b = 1/2$ introduces the most randomness and produces the best snare sounds.
    - b near 0 simply averages the samples, and produces string-like sounds where p controls the pitch. Note: doesn't work for constant or sine wavetables.
    - b near 1 produces wierd electric crash cymbal-like sounds where most of the pitches die out quickly. Note: doesn't work for constant or sine wavetables.

See Ch5_7_Physical_Modelling_Synthesis.mlx for code and to hear results.

Crash cymbal sounds

Choose values:

- $b > 0.98$,
- $p = 200 - -800$,
- **random wavetable**
- **decaying envelope**

See Ch5_7_Physical_Modelling_Synthesis.mlx for code and to hear results

Metallic plink sounds

Choose values:

- $b > 0.98$,
- $p = 5 - -50$,
- **random wavetable**
- **Envelope decay usually needed for $b = 1$**

See Ch5_7_Physical_Modelling_Synthesis.mlx for code and to hear results

And back to string sounds

Choose values:

- $b < 0.05$,
- $p = 20 - -400$,
- **random wavetable**
- **Envelope decay usually needed for** $b = 1$

See Ch5_7_Physical_Modelling_Synthesis.mlx for code and to hear results

Some more examples

See Ch5_7_Physical_Modelling_Synthesis.mlx for

- FULL MATLAB (Demo) EXAMPLE: Generating Guitar Chords Using the Karplus-Strong Algorithm
  - Playing a Note on an Open String
  - Playing a Note on a Fretted String
  - Playing Guitar Chords
  - Guitar Strumming