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:

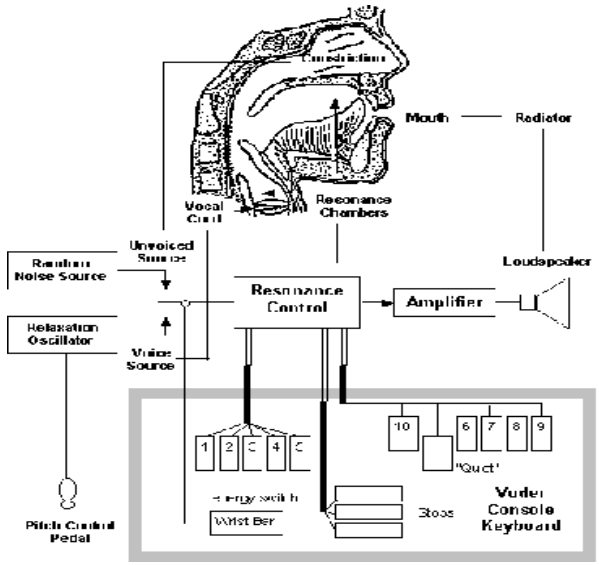
- Subtractive synthesis
- Additive synthesis
- FM (Frequency Modulation) Synthesis
- Sample-based synthesis
- Wavetable synthesis
- Granular Synthesis
- Physical Modelling

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).

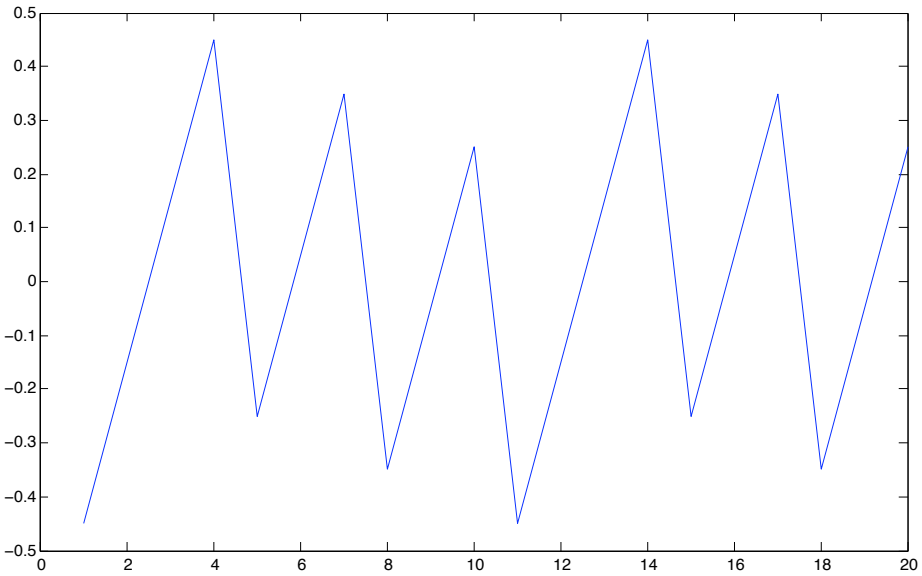
Popularised with Moog Synthesisers 1960-1970s

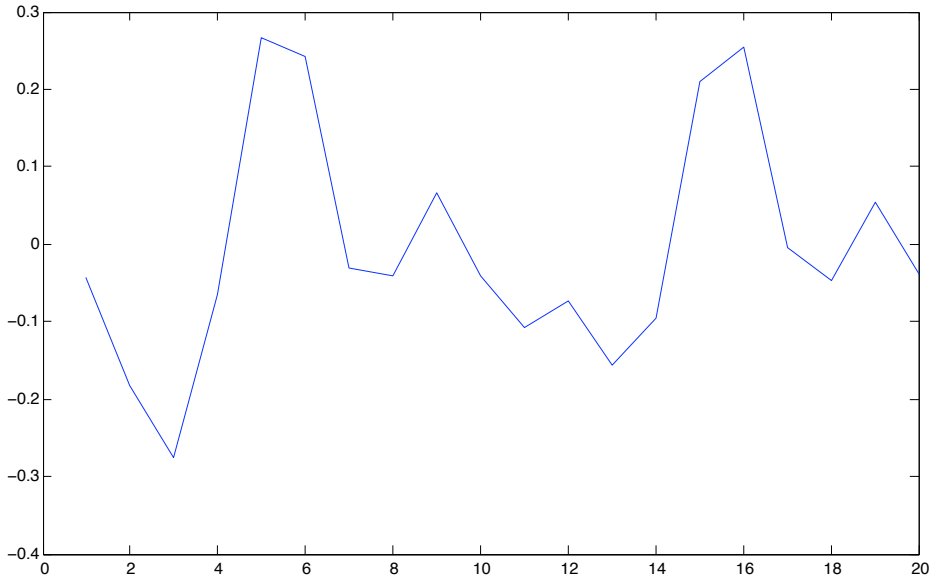


Subtractive synthesis: Simple Example

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.

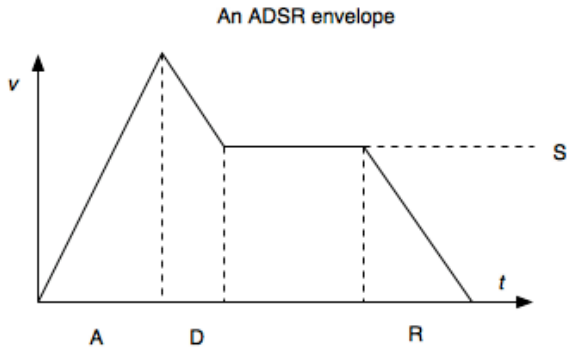
Further Processing: ADSR Envelope (Cont.)

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)

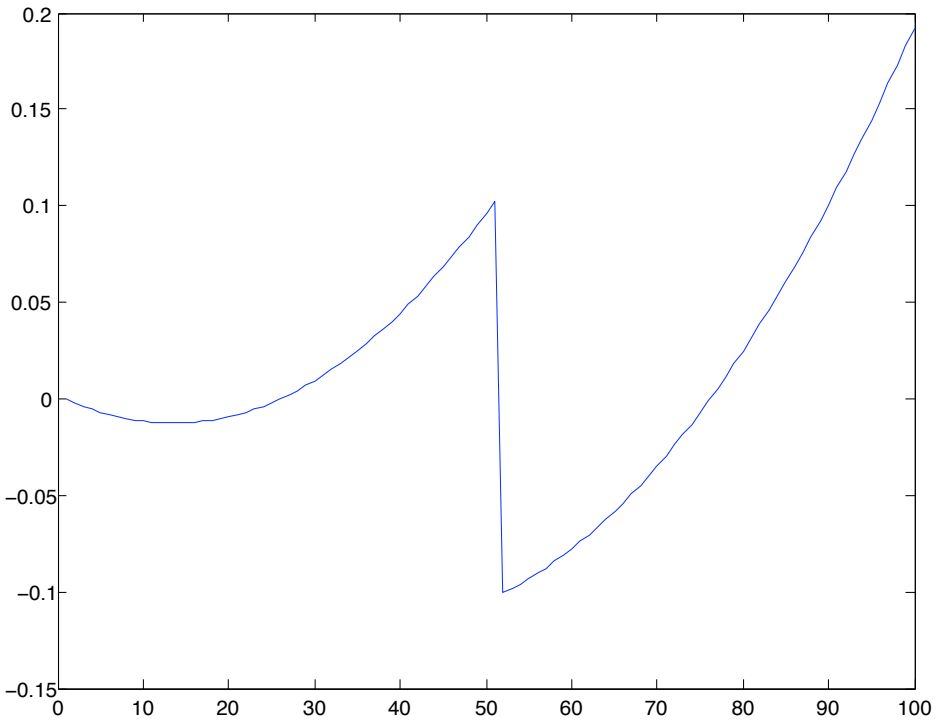
```
N = floor(dur*Fs);
n=0:N-1;
if (strcmp(type,'sine'))
  y = amp.*sin(2*pi*n*freq/Fs);

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:

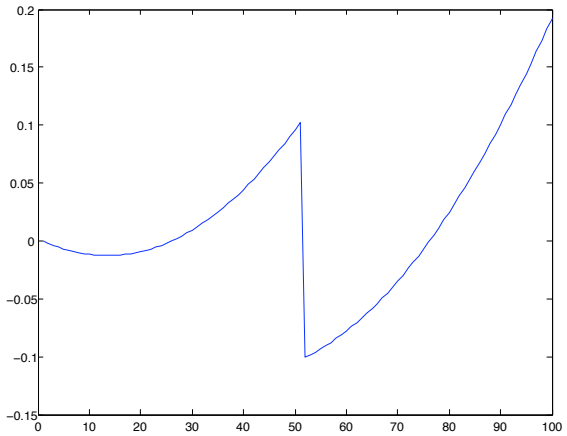


synth.m (Cont.)

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.