SON 2026-2027 lecture 5

3TC Audio Project @ INSA Lyon

Tanguy Risset

https://inria-emeraude.github.io/son/

Audio Processing Basics I

White Noise

- White noise is composed of an infinite number of harmonics all having the same level.
- The spectrum of white noise looks completely flat.
- White noise is produced by generating random numbers between -1 and 1. (see `Noise.cpp`)

```
Noise::Noise() :
randDiv(1.0/RAND_MAX){}

float Noise::tick(){
  return rand()*randDiv*2 - 1;
}
```

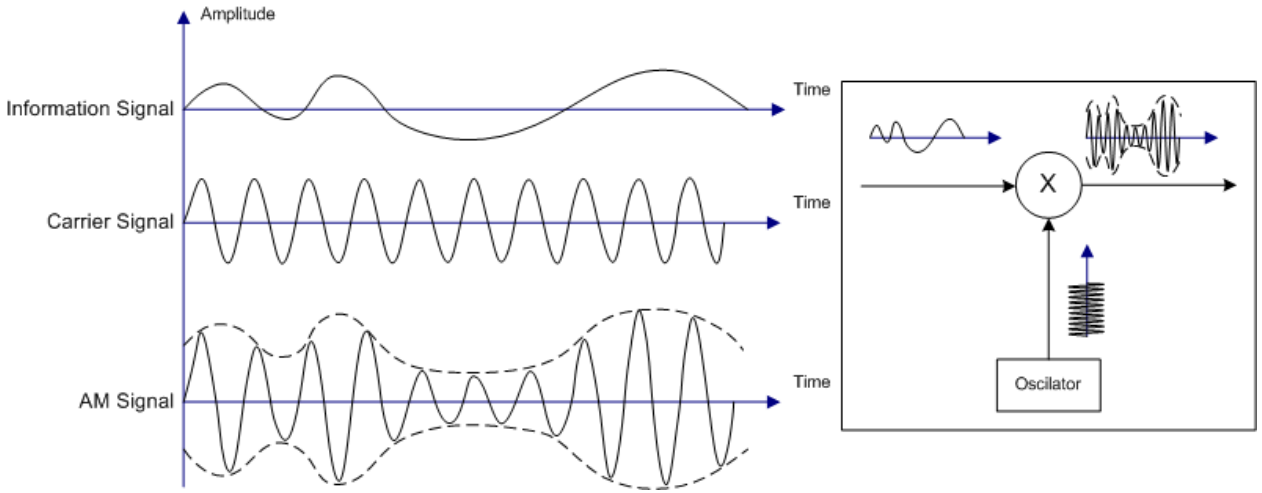
Wave Shape Synthesis

- Wave Shape synthesis is one of the most basic sound synthesis technique.
- It consists of using oscillators producing waveforms of different shapes to generate sound.
- The most standard wave shapes are:
- sine wave,
- square wave,
- triangle wave,
- sawtooth wave.

The `crazy-sine` example can be considered as "wave shape synthesis" in that regard.

Amplitude Modulation (AM) Synthesis

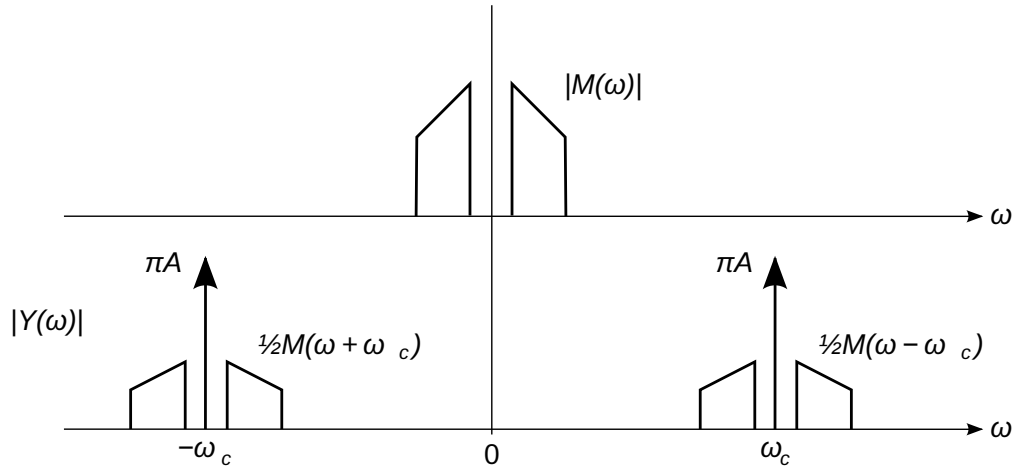
- Amplitude modulation synthesis consists of modulating the amplitude of a signal with another one.
- Sine waves are typically used for that (See `Am.cpp`):



Amplitude Modulation (Source: Wikipedia)

Amplitude Modulation (AM) Spectrum

- When the frequency of the modulator is low (bellow 20Hz), it creates a *tremolo* effect.
- However, above 20Hz two side bands (if sine waves are used) start appearing following this rule (see Julius Smith's website)



Amplitude Modulation Spectrum (Source: Wikipedia)

Amplitude Modulation in MyDsp

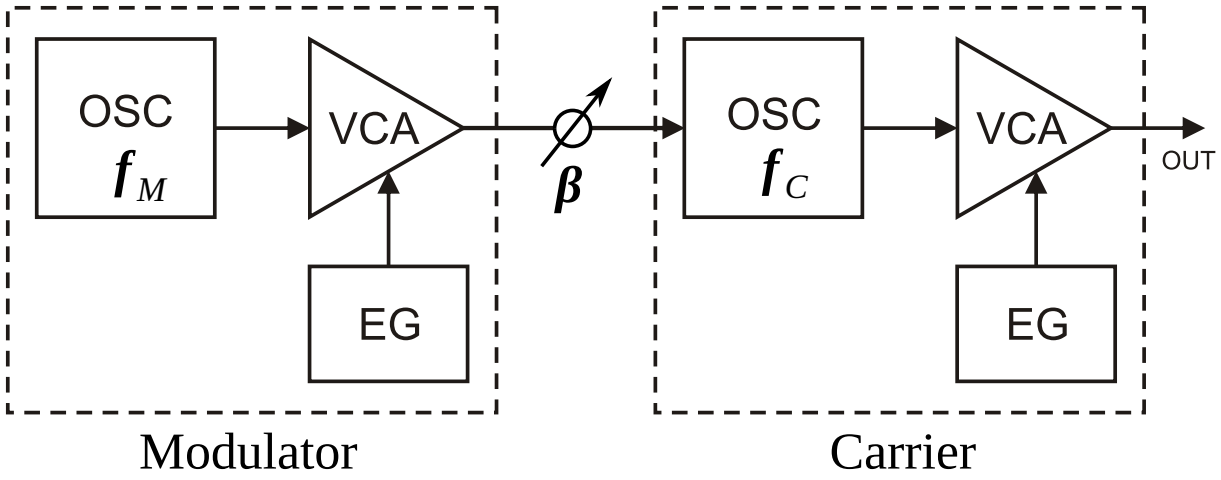
- The `am` example demonstrates a use case of an AM synthesizer.

```
float Am::tick(){
  int cIndex = cPhasor.tick()*SINE_TABLE_SIZE;
  int mIndex = mPhasor.tick()*SINE_TABLE_SIZE;
  float posMod = sineTable.tick(mIndex)*0.5 + 0.5;
  return sineTable.tick(cIndex)*(1 - posMod*modIndex)*gain;
}
```

- Again phasors are used instead of "complete" sine wave oscillators.
- The amplitude parameter of the modulating oscillator is called the *index of modulation*
- The frequency of the modulating oscillator is called *frequency of modulation*.
- The range of the modulating oscillator is adjusted to be {0,1} instead of {-1,1}.

Frequency Modulation (FM) Synthesis

- Frequency modulation synthesis consists of modulating the frequency of an oscillator with another one:
- `fm.cpp` provides a simple example of how an FM synthesizer can be implemented:



Frequency Modulation (Source: Wikipedia)

Frequency modulation theory

FM can be expressed as:

$$x(t) = A_c \sin[\omega_c t + \phi_c + A_m \sin(\omega_m t + \phi_m)]$$

where c denotes the carrier and m, the modulator.

- As for AM,
  - frequency and amplitude of modulating oscillator are called *frequency of modulation* and *modulation index*
- Unlike AM,
  - the value of the index of modulation can exceed 1 which will increase the number of sidebands.
  - FM is not limited to two sidebands and can have an infinite number of sidebands depending on the value of the index.
  - The mathematical rational behind this can be found on Julius Smith's website.

Frequency Modulation in MyDsp

The examples folder of the course repository hosts a simple Teensy program illustrating the use of FM.

```
float Fm::tick(){
  int mIndex = mPhasor.tick()*SINE_TABLE_SIZE;
  float modulator = sineTable.tick(mIndex);
  cPhasor.setFrequency(cFreq + modulator*modIndex);
  int cIndex = cPhasor.tick()*SINE_TABLE_SIZE;
  return sineTable.tick(cIndex)*gain;
}
```

- FM synthesis was discovered in the late 1960s by John Chowning at Stanford University in California.
- He's now considered as one of the funding fathers of music technology and computer music.
- FM completely revolutionized the world of music in the 1980s by allowing Yamaha to produce the first commercial digital synthesizers: the DX7 which met a huge success.

Simple Filter: One Zero

- Filters are at the basis of many audio effects such as Wah guitar pedals, etc.
- The most basic filter is what we call a "one zero" filter.
- The differential equation of a one zero filter can be expressed as:

$$y(n) = b_0 x(n) + b_1 x(n-1)$$

- $b_1 is called * feedforward coefficient*, it sets the zero of the pole$(-b_1/b_0$) b_0 can be discarded as it is equal to 1 in most cases.
- One zero filters can either be used as a lowpass if the value of b_1 is positive or as a highpass if b_1 is negative. - The frequency response of the filter which is be obtained with $H(e^{j\omega T}) = b_0 + b_1 e^{-j\omega T}$ can be visualized on Julius Smith's website.

One Zero in MyDsp

`OneZero.cpp` implements a one zero filter:

```
float OneZero::tick(float input){
  float output = input + del*b1;
  del = input;
  return output*0.5;
}
```

Note that we multiply the output by 0.5 to normalize the output gain.

The `filtered-noise` example program for the Teensy demonstrates the use of `OneZero.cpp` by feeding white noise in it.

Biquad Filter

- Combining zeros (like the one zero filter) with poles gives much sharper and more resonant filters.
- The most widely used building block for audio filtering is the *biquad* (two poles, two zeros), whose difference equation is:

$$y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) - a_1 y(n-1) - a_2 y(n-2)$$

- The b_i coefficients set the zeros (feed forward), the a_i coefficients set the poles (feedback).
- Poles must stay inside the unit circle ($|a_2| < 1$) otherwise the filter becomes unstable and the output blows up.
- Full derivation and pole/zero analysis on Julius Smith's website.

Biquad: Why It Matters

- A single biquad, by only changing its 5 coefficients, can implement lowpass, highpass, bandpass, notch, peak (EQ) and shelving filters.
- This makes it the standard building block of:
  - parametric/graphic equalizers,
  - resonant filters used in synthesizers (e.g. the classic Moog-style filter sweep),
- Cascading several biquads ("second-order sections") is also the standard way to build higher order filters while keeping numerical stability.
- Designig resonant filter is usually done using bandwidth, gain and Q (resonance).
- We will see on next course how to compute the biquad filters coefficient from these three parameter (see the RBJ "Audio EQ Cookbook")
-

Exercises

- LFO: Low Frequency Oscillator
- Towards the DX7
- Smoothing

LFO: Low Frequency Oscillator

An LFO is an oscillator whose frequency is below the human hearing range (20 Hz). LFOs are typically used to create vibrato. In that case, the frequency of the LFO is usually set to 6 Hz.

Modify the crazy-saw example so that notes are played slower (1 per second) and that some vibrato is added to the generated sound.

Towards the DX7

The DX7 carried out frequency modulation over a total of six oscillators that could be patched in different ways. So FM is not limited to two oscillators... Try to implement an FM synthesizer involving 3 oscillators instead of one. They should be connected in series: 3 -> 2 -> 1.

Smoothing

- In most cases, DSP parameters are executed at control rate.
- changing the value of a DSP parameter will often result in a "click"/discontinuity.
- A common way to prevent this from happening is to interpolate between the values of the parameter using a "leaky integrator." In signal processing, this can be easily implemented using a normalized one pole lowpass filter:

$$y(n) = (1 - s)x(n) + sy(n - 1)$$

where s is the value of the pole and is typically set to 0.999 for good results.

Modify the crazy-saw example by "smoothing" (Use the `Smooth` class) the value of the frequency parameter by implementing the filter above with $s = 0.999$. Then slow down the rate at which frequency is being changed so that only two new values are generated per second. The result should sound quite funny :).

Smoothing Potentiometer Values

Try to use the smoothing function that you implemented in the previous step to smooth sensor values coming from a potential potentiometer controlling some parameter of one of the Teensy examples. The main idea is to get rid of sound artifacts when making abrupt changes in potentiometers.

End of Lecture 5