

# Subtractive Synthesis

The model for subtractive synthesis is complementary to the previously seen additive techniques



Most of the dedicated out-board synthesizers are generally based on this synthesis model. They usually offer a limited number of sources, like fixed waveshapes or sampled sound

In software synthesis, the model remains the same. The choices for sources and modifiers are more varied (in fact they can be of any description).

## Sources

Any number of:

- **oscillators**, with any waveshape
- **band-limited pulse**: creates waves composed of  $N$  harmonics, all with the same relative amplitude, the *buzz* generator.
- **noise generators** (three basic types: rand, randi, randh)
- **input sound**, from disk (soundfile) or from realtime input (in, soundin, diskin, etc.).
- **other** signal generators and synthesis algorithms.

## Band Limited Pulse (blp)

Band-limited pulses can be generated with the **buzz** opcode in `csound`:

```
ar      buzz      xamp, xcps, knh, ifn[, iphs]
```

**xamp** - signal amplitude

**xcps** - frequency

**knh** - number of harmonics

**ifn** - function table number, containing a **sine** wave

# Filter

Filters shape the input sound, modifying its spectrum.

The idea is that the filter lets parts of the sound pass without change, while blocking others.

Three types of filter are normally found:

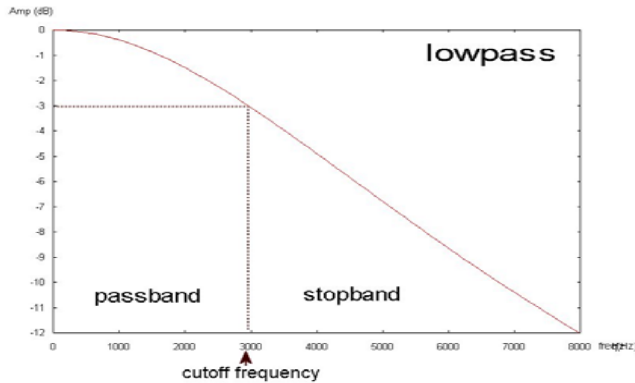
1. **Low-pass:** lets low frequencies pass while cutting the high ones
2. **High-pass:** lets high frequencies pass while cutting the low ones
3. **Band-pass:** lets a certain band of frequencies pass, cutting the ones outside it.

## **Filter parameters**

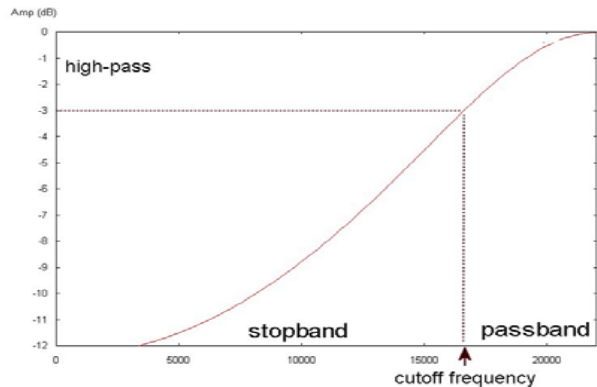
**Frequency:** for low-pass and high-pass filters, this indicates the **cut-off frequency**, frequency above/below which the filter starts to be effective.

For a band-pass filter, this indicates the **centre frequency** at the centre of the pass-band.

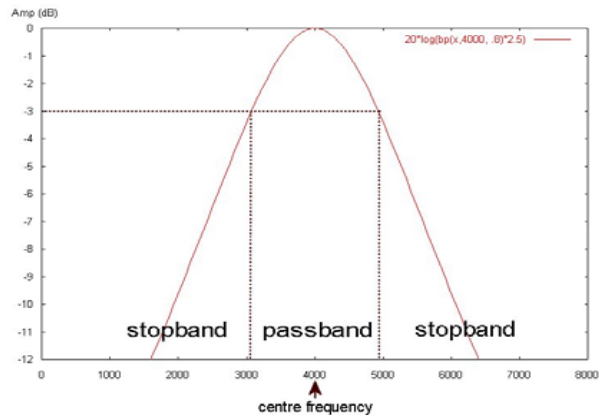
**Bandwidth:** (only band-pass) the width of the pass-band, how much is taken out or left in the original sound



*Low-pass filter*: passes frequencies below the cutoff frequency and cuts frequencies above it



*High-pass*: stops frequencies below the cutoff frequency and passes frequencies above it.



*Band-pass*: passes frequencies in the band defined by a bandwidth (BW) and centre frequency (all frequencies  $cf \pm \frac{1}{2}BW$  ).

## Filter UGs

In **csound** there is a whole family of different types of filters, each one with different qualities.

Most of them will fit one of the four categories of filters that we've already seen: lowpass, hipass, bandpass, etc.

The basic type of filter present in **csound** and most computer music languages is the standard 2nd-order bandpass filter called **reson**.

In addition to this, *butterworth* filters are also present in **csound**, providing a quality filtering in band, low and high-pass shapes, as well as band-reject.

## Reson

The **reson** filter is a standard multi-purpose filter found in most computer music languages.

```
ar    reson asig, kcf, kbw [, iscl]
```

**asig** - signal input

**kcf** - filter centre frequency

**kbw** - filter bandwidth

**iscl** - output amplitude scaling code: 0 - *no scaling* (default)

1 - scale gain at centre freq to 1 (*normalise*, for periodic *srcs*),

2 - scale the gain so that the overall RMS amp is 1 (for *noise srcs*)

## The Q

Rather than specifying the bandwidth directly, is often common to make it dependent on the centre frequency. For this reasons, a **quality factor Q** , is defined as:

$$Q = \text{freq} / \text{BW}$$

The  $Q$  value determines how much is filtered out: high values will make the filtering more selective (narrow bandwidths) and vice-versa.

## **Q and Musical Intervals**

The Q value can be related to musical intervals, since both are fixed frequency ratios.

For instance, the Q of 1.5 is equivalent to the bandwidth of an octave (2:1), e.g.:

$$\mathbf{75 - 150 \text{ Hz, } *bandwidth*: 75 \text{ Hz, } *centre freq*: 112.5 \text{ Hz}}$$
$$\mathbf{Q = 112.5 / 75 = 1.5}$$

The Q for a perfect 5th (3:2) is 2.5:

$$\mathbf{150 - 225, } *bw*: 75 \text{ Hz, } *centre freq*: 187.5 \text{ Hz}$$
$$\mathbf{Q = 187.5 / 75 = 2.5}$$

## Balancing signals

If the filter output is left unscaled, huge variations of signal can occur, causing clipping and distortion. For this reason an opcode that scales a signal with relation to another is present:

```
ar balance asig, acomp
```

The input signal **asig** is brought to the same RMS level as the signal **acom** (the comparative signal).

Balance can be used as an *amplitude follower*: the input signal amplitude is shaped to match that of the comparator signal.

## Example

```
asrc buzz 10000, 440, sr/440, 1 ; band-limited pulse train
aflt reson asrc, 1000, 700 ; sent through a filter
aflt2 reson aflt, 1200, 100 ; and another
afin balance aflt2, asrc ; then balanced with source
out afin
```

This example shows a cascade (or series) connection of filters. This is one of the two ways of connecting multiple filters. The other type of connection is based on parallel filters.

## Time Response

Filters also have what we can describe as their *time response*, or how they react to quick changes in amplitude of the input sound.

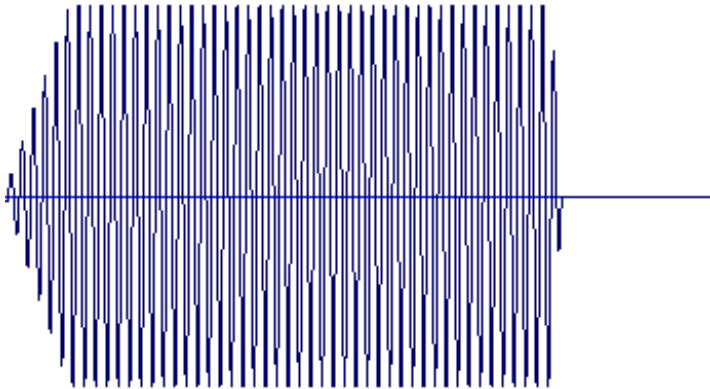
This is also described as the filter *transient* response. It relates to how quickly a filter settles into its steady-state behaviour, after some a change in the input.

The time response is inversely related to the bandwidth: the narrower the filter, the slower it will react to changes in amplitude of the input.

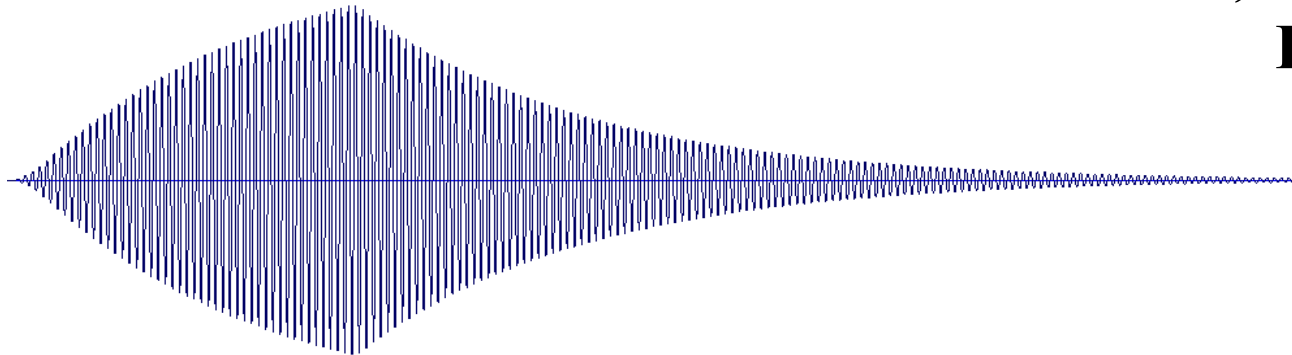
# example



**input: 1000 Hz sine**



**output: filtered by a  
reson, CF:1000Hz  
BW: 10 Hz**



## Frequency Response versus Time Response

In any filter, the more selective in terms of frequency we are (higher  $Q$ s), the worse the time response.

The typical example is that of a filter with a very narrow band, where the envelope of the input sound will be distorted:

- (a) the attack and decay will be slowed down
- (b) after the original sound has stopped, a ringing may be heard in the output.

This is why very selective filters ('brickwall') are problematic.

# Subtractive Synthesis Designs

## 1. A simple LP design

Using a BP reson, a Low Pass filter can be obtained by setting the center frequency (CF) to 0 and using the bandwidth (BW) to control the cutoff frequency.

The filter will have half of its passband on the positive side and half on the negative. The upper edge of the passband will be equivalent to the LP cutoff frequency

However, due to approx.. errors, the cutoff frequency will be  **$0.707 * BW$** .

The reson parameters will be:  **$CF = 0$**      **$BW = \text{cutoff\_freq}/0.707$**

## Csound code

The source is a blp with all harmonics up to the Nyquist  
**((sr/2)/fund** *is the number of harmonics needed*).

```
instr 1

iamp = p4
ifund = p5
icutfr = p6

asig      buzz  iamp, ifund, (sr/2)/ifund, 1
aflt      reson asig, 0, icutfr/0.707, 1
           out      aflt

endin
```

# Dynamic Spectra

Most of the interesting sounds around us are based on spectra that changes throughout time.

Filters can have time-varying BWs & CFs, creating dynamic spectra.

For instance, the previous instrument can be adapted to have a change in spectral brightness as its intensity changes.

In this case the bandwidth can be controlled by the same envelope that controls amplitude.

# Csound code

```
instr 1

iamp = p4
ifund = p5
iscal = .1 ; scaling factor(cutoff freq is 10% of max
           ; amplitude value)

kenv    linen iamp, 0.03, p3, 0.1
asig    buzz  kenv, ifund, 22500/ifund, 1
kcutfr  = kenv*iscal
aflt    reson asig, 0, kcutfr/0.707, 1
        out    aflt

endin
```

# **Multiple filter arrangements**

## **1. Parallel filterbank**

filters arranged in parallel. The same input signal is fed to all filter inputs. Their outputs are mixed together. The overall amplitude response is the sum of individual responses.

## **2. Cascade (series) filterbank**

filters are fed the output signal of the previous filters in the series. The input signal is fed to the first filter and the output is taken from the last. The overall amplitude response is the product of individual responses.

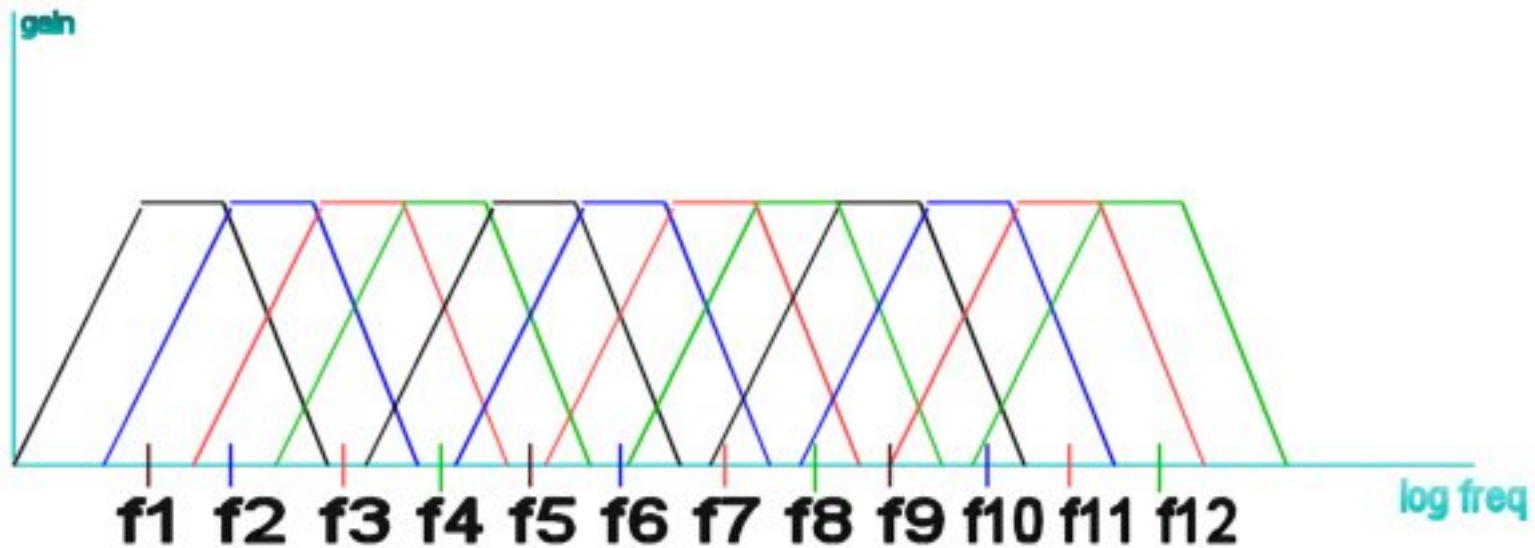
## Parallel Connections

When connecting BP filters in parallel, we are actually creating a more complex filter, with more than one pass-band regions.

This can be used to create complex spectral envelope shapes, with many peaks.

A typical example of the use of a bank of parallel filters is found in **graphic equalisers**. These employ multiple BP filters, with constant-Q, spaced at even intervals (e.g. 1/3 octave, etc.). Their output gain will determine how much energy is passed at each band. Adjacent filters usually have some pass-band overlap, so that there is no 'holes' in the spectrum.

# Graphic Equaliser



This is an example of a 12-band Graphic Eq, with 12 overlapping band-pass filters, spaced at even frequency intervals.

## Series Connection

A bank of filters connected in series is usually employed in two cases, to:

(a) create a more selective filter: using two BP filters (2nd order) in series with the same CF & BW, we make up a 4th order filter, which will (i) have a steeper roll-off (ii) have a narrower BW (at -3dB).

(b) create a filter with a wider passband, but with a steeper rolloff: using two filters with CF that are close to each other and overlapping passbands.

## **Roll-off and Bandwidth**

The overall amplitude response in the series connection is the product of the individual responses.

For this reason, the roll-off of two identical BP filters in series will be steeper, as we multiply each the two responses point by point.

At the points where the individual attenuation is 3dB, the combined attenuation is 6dB. So the new -3dB BW of the series connection will be the individual -1.5dB BW of each filter, which is narrower.

So, the resulting 4th-order filter has a steeper rolloff and a narrower bandwidth.

## Balancing the signal

When connecting filters in series, especially when the filters are of different specs (CF & BW), the output can be very low in intensity.

This is because we are filtering a signal that has already been filtered, so part of its spectrum has already been attenuated. If we attenuate the other frequency ranges that have been left, we might end up with very little signal.

The **balance** unit is needed in such situations, so that we can bring the signal to the original level before filtering.

## Csound code examples

```
instr 1 ; parallel
iq = 25
asig      in
aflt1     reson  asig, 700, 700/iq, 1
aflt2     reson  asig, 1800, 1800/iq, 1
          out      aflt1+aflt2
endin
```

```
instr 1 ; series
iq = 25
asig      in
aflt1     reson  asig, 700, 700/iq, 1
aflt2     reson  aflt1, 1800, 1800/iq, 1
abal      balance aflt2, asig
          out      abal
endin
```

## **Parallel filterbanks and formants**

A set of formants for a specific timbre (say any vowel) can be created using parallel filters, each one for a specific formant region.

In this case, the formant parameters of centre frequency and bandwidth can be used directly in the band-pass filters.

The amplitude of each formant is used to scale (multiply) the output of each filter.

Example: three formant regions: 600, 1100, 2500; with 80, 100 and 300Hz BW, respectively, at 0, -9 and -18 dB

```
afor1  reson  asig, 600, 80, 1          ; asig is the signal input
afor2  reson  asig, 1100, 100, 1
afor3  reson  asig, 2500, 300, 1
amix   = afor1 + afor2*ampdb(-9) + afor3*ampdb(-18)
out    amix
```

This instrument uses **ampdb(...)** which converts a value in the dB ratio to an amplitude multiplier (to scale a signal). For instance, **ampdb(-6)** will convert to a value close to 0.5.

# The Vocoder

Here we split the spectrum in several bands, analyse the amount of energy on each of them, and then impose that spectral envelope on another sound (the ‘excitation source’). Two filter elements are used:

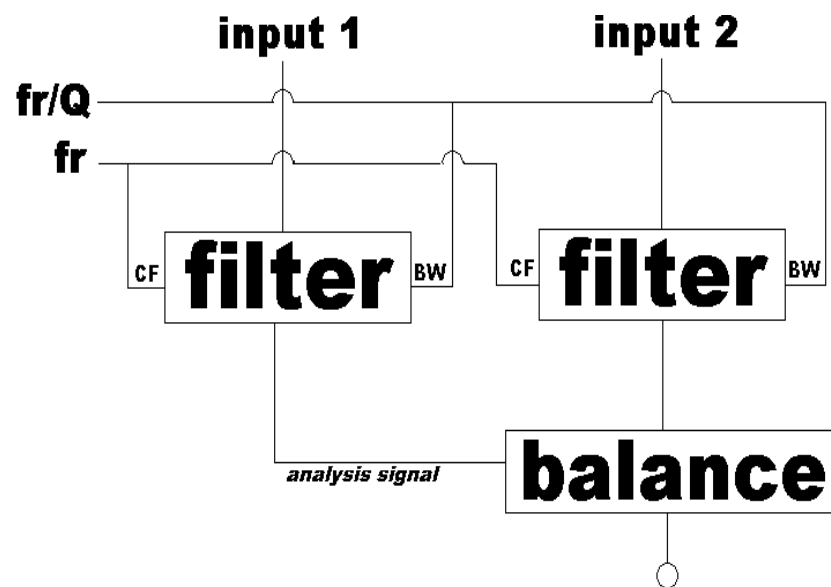
(a) **analysis filters**: used to break the input spectrum into bands and find the energy on each band (its RMS amplitude).

(b) **synthesis filters**: used to filter out a source and impose the analysed spectral envelope. This part of the design is just like a graphic EQ.

The RMS amplitude levels of each analysed band are imposed on the filtered source signal using **balance**.

## Band design

Each analysis/synthesis band will have the following design:



**input 1** is the *analysis signal*, **input 2** the *source sound*.

*Each band* will have a different *CF* and a *BW* (determined by the overall  $Q$ ). The **balance** matches the amplitude of the analysis signal with the filtered signal.

## Csound code

This is a sample code for **one** vocoder band, as described before:

```
anall reson ainput1, icf1, icf1/iq  
asyn1 reson ainput2, icf1, icf1/iq  
abnd1 balance asyn1, anall
```

In the code above, *ainput1* and *ainput2* are the analysed signal and the excitation source, respectively.

The centre frequency for the band is *icf1* and the filter  $Q$  is *iq*.

## Bands and Sources

The number of bands will determine how accurate the result is. The more bands, the better analysis, but with narrower filters, we will have a loss in the time response. In addition, more computation will be needed to calculate the output when more filters are used.

As for the excitation source, any complex (component-rich) source will be suitable, pitched or un-pitched.

The vocoder design is an example of cross-synthesis: imposing the spectral envelope of one sound into another.

You can think of it as an *automated graphic EQ*, which uses an input sound to control the gain of each EQ band.

## A closer look at filters

Filters are implemented by combining signals with their delayed copies, in different ways. Two basic families of filters exist:

(1) Filters that use delayed copies of their input, called feedforward or finite impulse response (FIR).

(2) Filters that use delayed outputs, called feedback or infinite impulse response (IIR). They can also include feedforward elements. Resonators and Butterworth filters are IIRs.

In digital filters, delays will be as small as 1 sample. The *order* of the filter is determined by the order of the delay used. If a filter uses a max of 2-sample delays, it is classed as *2<sup>nd</sup> order*.

## Filter Equations

Filters are defined by their equations. These will show:

- (a) the delays used in the filter.
- (b) the coefficients (gain multipliers) associated with each delay.

Example:

$$y(n) = ax(n) + a_1x(n-2) - b_1y(n-1) - b_2y(n-2)$$

$n$  is the current sample of a signal;  $y(n)$  is the output and  $x(n)$  the input.

$a_1$  is the coefficient associated with a 2-sample delay of the input and  $b_1 / b_2$  are the coefficients associated with 1- and 2-sample delays of the filter output.

## Coefficients and Filter Frequency Response

From its coefficients, we can determine the filter frequency response, and vice-versa.

The frequency response is how a filter alters an input signal, in terms of its *amplitude* and *phase* at *different frequencies*.

The *amplitude response* is the part of the frequency response that has to do with boosting or attenuating the different input frequencies.

The *phase response* determines the timing delays imposed on different frequencies. It can be linear (the same phase change at all frequencies) or non-linear.

## **Filter Implementation**

**IIR** filters are generally simple to implement, but complex to design. One way round this is that they are generally offered in pre-packed formats, ie. *resonators*, *butterworths*, *ellipticals*, etc. Their cutoff frequencies and bandwidths can be made time-varying, which is an important characteristic for musical applications.

**FIR** filters are simpler to design and can offer linear-phase characteristics, so they are often preferred by engineers. Implementing them is also easy, but depending on the order of the filter, they can be computationally intensive and are not time-varying.