

Subtractive synthesis designs: the vocoder

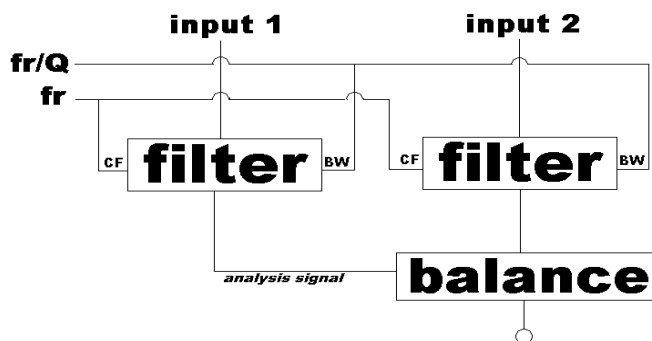
The vocoder design presented here is modelled on the analog, filter-based, vocoders found in some commercial synthesizers of the 70s. These in turn were a modification of the original channel vocoder developed at the Bell Labs in the 40s and 50s. A digital, much more advanced version of this tool, is the so-called Phase Vocoder, which should not be confused with our design here. The Phase Vocoder is a frequency-domain analysis/resynthesis tool, which will be studied later in the course. The vocoder design here is a time-domain, filterbank based, algorithm. It extracts the spectral envelope of a sound and imposes it on another sound, performing something we call ‘cross-synthesis’.

We can extract, for instance the formants of a vocal sound and impose them on the spectrum of a synthesised source (e.g. a buzz).

The idea behind this design is that we are able to use a bank of parallel filters to extract the spectral envelope, or the overall energy at different frequency regions, of a sound (we call this the ‘analysis filterbank’). Each filter is a band pass (BP) resonator, tuned to a particular frequency FR and with a certain bandwidth BW . In order to make the bandwidth a constant-size interval, we can make it dependent on the frequency by using the Q factor (FR/BW). To extract the spectral envelope all we need is to filter the sound using the analysis filterbank. The amount of energy at the output of each filter is going to be proportional to the energy of the sound at that particular frequency band.

Once we have a signal energy to track, we can then use a similar filterbank (the ‘synthesis filterbank’) to analyse another sound and control the output of each filter with the amplitude of the corresponding analysis filter. The energy of each band of the analysed sound will be imposed on each band of the synthesised sound. We can use a **balance** opcode to track and control the amplitude of each pair of filters.

A vocoder band will look like this (the analysed sound comes on input 1):



On csound we'll have:

```
analysis_filter1  reson  ain1, if1, if1/iq, 1
asynthesis_filter1  reson  ain2, if1, if1/iq, 1
aoutput1         balance  asynthesis_filter1, analysis_filter1
```

A vocoder can typically have 12 – 24 of these bands, each one tuned to a different frequency and often sharing the same Q . Sometimes these bands are spaced evenly (linearly), other times logarithmically.

Given the fact that our perception is logarithmic, the second spacing method will probably result in a better performance. A combination of the two is also possible.

Here's the complete instrument. It uses either a **buzz** or a **rand** as the source for the synthesised sound. The analysis sound comes from **in** (and **-i ...**). The score p-fields are: **p4** (buzz/rand amplitude), **p5** (buzz fundamental), **p6** (Q factor) and **p7** (source type, 1 for buzz, anything else for rand).

The choice of synthesis source signal is implemented using a simple control flow statement in csound:

```
if isrc == 1 goto blp
```

which means “if isrc is 1 go to the label ‘blp’”, else just continue reading, which takes us to the line defining the noise generator:

```
asig rand p4  
goto input
```

the ‘goto input’ line tells it to jump to the label ‘input’. Labels are lines starting with a keyword followed by a semicolon, such as:

```
blp:  
ifreq = p5  
asig buzz iamp, ifreq, 50, 1
```

and

```
input:  
ain in
```

So the control-flow statement decides whether we'll use rand or buzz, depending on the value of `isrc` (which is taken from `p6`). Completing the instrument, I use a `linseg` to avoid clicks at the beginning and ending of the sound, by fading in and out.

```
; 20-band vocoder  
; Victor Lazzarini, 1998
```

```
sr= 44100  
kr = 4410  
ksmps = 10  
nchnls= 1
```

```
instr 1
```

```
;filter center frequencies  
if1 = 80  
if2 = 160  
if3 = 200  
if4 = 320  
if5 = 380  
if6 = 490  
if7 = 640
```

```

if8 = 760
if9 = 980
if10 = 1040
if11 = 1280
if12 = 1400
if13 = 1600
if14 = 2080
if15 = 2300
if16 = 2560
if17 = 3200
if18 = 4160
if19 = 5120
if20 = 6400

;filter q
iq = p6

;source
isrc = p7

;amplitude
iamp = p4

if isrc == 1 goto blp
asig rand p4
goto input

blp:
ifreq = p5
asig buzz iamp, ifreq, 50, 1

input:
ain in

;filters

analysis_filter1 reson ain, if1, if1/iq, 1
asynthesis_filter1 reson asig, if1, if1/iq, 1
aoutput1 balance asynthesis_filter1, analysis_filter1

analysis_filter2 reson ain, if2, if2/iq, 1
asynthesis_filter2 reson asig, if2, if2/iq, 1
aoutput2 balance asynthesis_filter2, analysis_filter2

analysis_filter3 reson ain, if3, if3/iq, 1
asynthesis_filter3 reson asig, if3, if3/iq, 1
aoutput3 balance asynthesis_filter3, analysis_filter3

analysis_filter4 reson ain, if4, if4/iq, 1
asynthesis_filter4 reson asig, if4, if4/iq, 1
aoutput4 balance asynthesis_filter4, analysis_filter4

analysis_filter5 reson ain, if5, if5/iq, 1
asynthesis_filter5 reson asig, if5, if5/iq, 1
aoutput5 balance asynthesis_filter5, analysis_filter5

analysis_filter6 reson ain, if6, if6/iq, 1

```

```

asynthesis_filter6 reson asig, if6, if6/iq, 1
aoutput6 balance asynthesis_filter6, analysis_filter6

analysis_filter7 reson ain, if7, if7/iq, 1
asynthesis_filter7 reson asig, if7, if7/iq, 1
aoutput7 balance asynthesis_filter7, analysis_filter7

analysis_filter8 reson ain, if8, if8/iq, 1
asynthesis_filter8 reson asig, if8, if8/iq, 1
aoutput8 balance asynthesis_filter8, analysis_filter8

analysis_filter9 reson ain, if9, if9/iq, 1
asynthesis_filter9 reson asig, if9, if9/iq, 1
aoutput9 balance asynthesis_filter9, analysis_filter9

analysis_filter10 reson ain, if10, if10/iq, 1
asynthesis_filter10 reson asig, if10, if10/iq, 1
aoutput10 balance asynthesis_filter10, analysis_filter10

analysis_filter11 reson ain, if11, if11/iq, 1
asynthesis_filter11 reson asig, if11, if11/iq, 1
aoutput11 balance asynthesis_filter11, analysis_filter11

analysis_filter12 reson ain, if12, if12/iq, 1
asynthesis_filter12 reson asig, if12, if12/iq, 1
aoutput12 balance asynthesis_filter12, analysis_filter12

analysis_filter13 reson ain, if13, if13/iq, 1
asynthesis_filter13 reson asig, if13, if13/iq, 1
aoutput13 balance asynthesis_filter13, analysis_filter13

analysis_filter14 reson ain, if14, if14/iq, 1
asynthesis_filter14 reson asig, if14, if14/iq, 1
aoutput14 balance asynthesis_filter14, analysis_filter14

analysis_filter15 reson ain, if15, if15/iq, 1
asynthesis_filter15 reson asig, if15, if15/iq, 1
aoutput15 balance asynthesis_filter15, analysis_filter15

analysis_filter16 reson ain, if16, if16/iq, 1
asynthesis_filter16 reson asig, if16, if16/iq, 1
aoutput16 balance asynthesis_filter16, analysis_filter16

analysis_filter17 reson ain, if17, if17/iq, 1
asynthesis_filter17 reson asig, if17, if17/iq, 1
aoutput17 balance asynthesis_filter17, analysis_filter17

analysis_filter18 reson ain, if18, if18/iq, 1
asynthesis_filter18 reson asig, if18, if18/iq, 1
aoutput18 balance asynthesis_filter18, analysis_filter18

analysis_filter19 reson ain, if19, if19/iq, 1
asynthesis_filter19 reson asig, if19, if19/iq, 1
aoutput19 balance asynthesis_filter19, analysis_filter19

analysis_filter20 reson ain, if20, if20/iq, 1
asynthesis_filter20 reson asig, if20, if20/iq, 1

```

```
aoutput20          balance asynthesis_filter20, analysis_filter20

amix1 = aoutput1 + aoutput2 + aoutput3 + aoutput 4
amix2 = aoutput5 + aoutput6 + aoutput7 + aoutput8 + aoutput9 + aoutput10
amix3 = aoutput11 + aoutput12 + aoutput13 + aoutput 14
amix4 = aoutput15 + aoutput16 + aoutput17 + aoutput18 + aoutput19 + aoutput20

kenv linseg 0, .1, 1, p3-.2, 1, .1, 0 ; trim the ends

      out (amix1+amix2+amix3+amix4)*kenv

endin
```