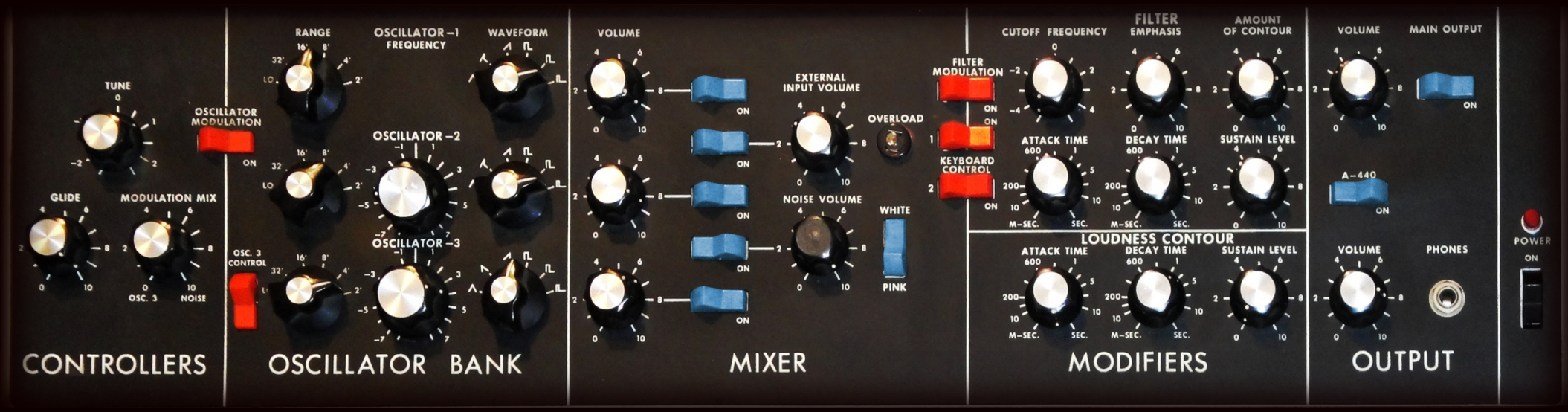
Lecture 4:
Subtractive Synthesis



ADDITIVE vs. SUBTRACTIVE SYNTHESIS

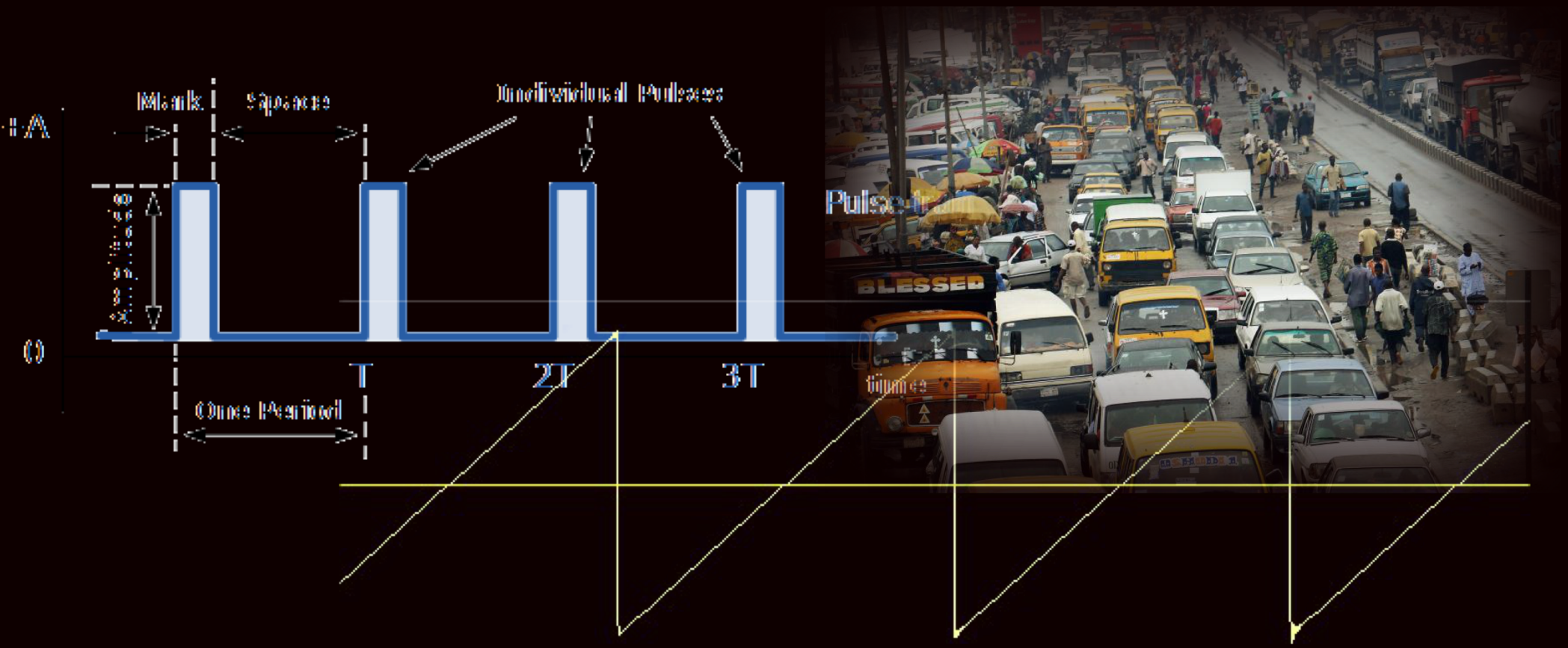
Additive Synthesis	Subtractive Synthesis
Powerful	Creation of precise timbres more difficult
Many input parameters	Fewer input parameters
User modulation difficult	User modulation easier
Potentially CPU intensive	CPU efficient
Creation of rich sounds requires much work	Rich sounds are created easily

SUBTRACTIVE SYNTHESIS DEFINITION

- Additive synthesis:
    - Based on the notion of building up complex sounds from simple ‘atomic’ sound elements.
- Subtractive synthesis:
    - Takes the opposite approach by beginning with a complex sound and reducing and refining it through filtering.

RICH SOUND SOURCES

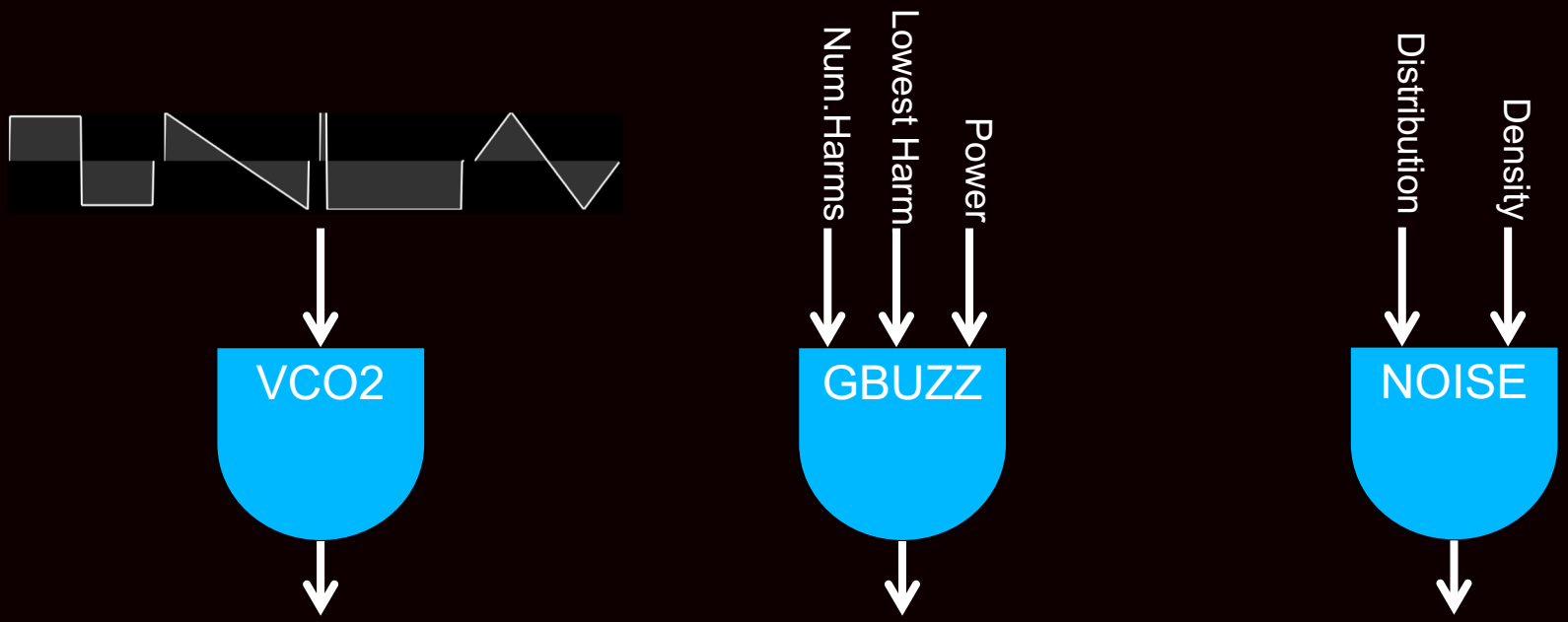
- Our rich sound source for subtractive synthesis could be a sawtooth waveform, a pulse waveform, a square waveform, generated noise, a sound sample of the sea, someone speaking, the traffic...



- A sine tone or a tuning fork are <u>not</u> rich sound sources.

RICH SOUND SOURCES IN CSOUND

- **`vco2, squinewave`**
  - Useful for generating band-limited basic waveshapes: sawtooth, square, triangle, pulse etc. (example 1 & 2)
- **`gbuzz`**
  - Useful for creating dynamic spectra based on stacks of harmonically related cosine waves. (example 3)
- **`noise, pinkish, pinker, dust2, gausstrig`...**
  - Useful for creating noisy source sounds. (example 4)



VCO2

- Opcode format

```
aSig  vco2  kAmp, kCPS, iType, kPW
```

```
; sawtooth waveform
aSig  vco2   0.2, 220
```

- **iType = 0** produces a sawtooth waveform but this is also the default, so can be omitted.

```
; square waveform
aSig  vco2   0.2, 220, 2, 0.5
```

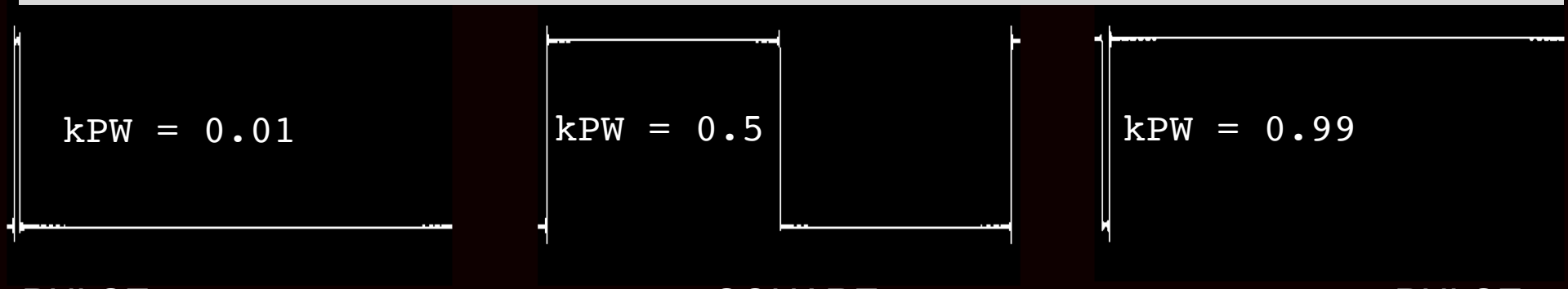
```
; triangle waveform
aSig  vco2   0.2, 220, 4, 0.5
```

```
; pulse waveform
aSig  vco2   0.2, 220, 2, 0.05
```

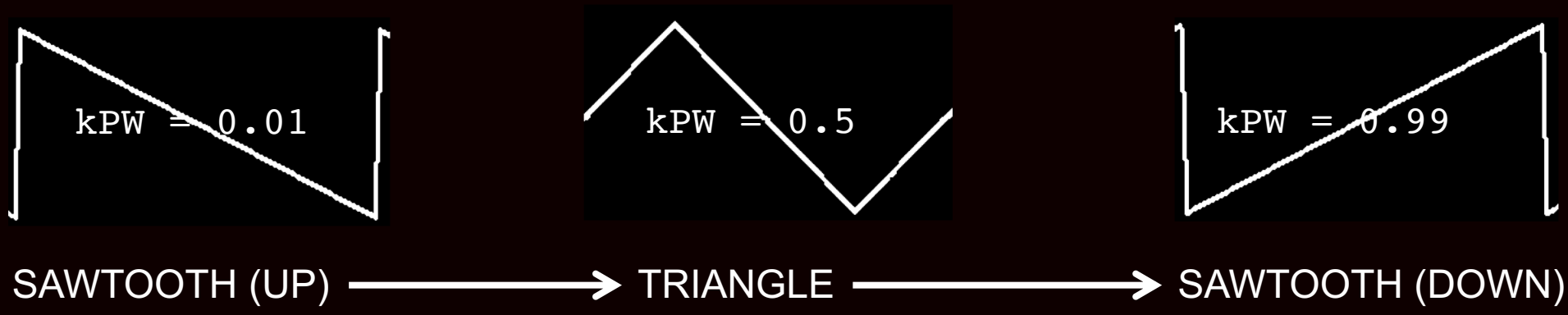
- See example 1.

VCO2 – PULSE-WIDTH MODULATION

```
; type = 2
kPW   line   0.01, p3, 0.99
aSig  vco2   0.2, 220, 2, kPW
```




```
; type = 4
kPW   line   0.01, p3, 0.99
aSig  vco2   0.2, 220, 4, kPW
```

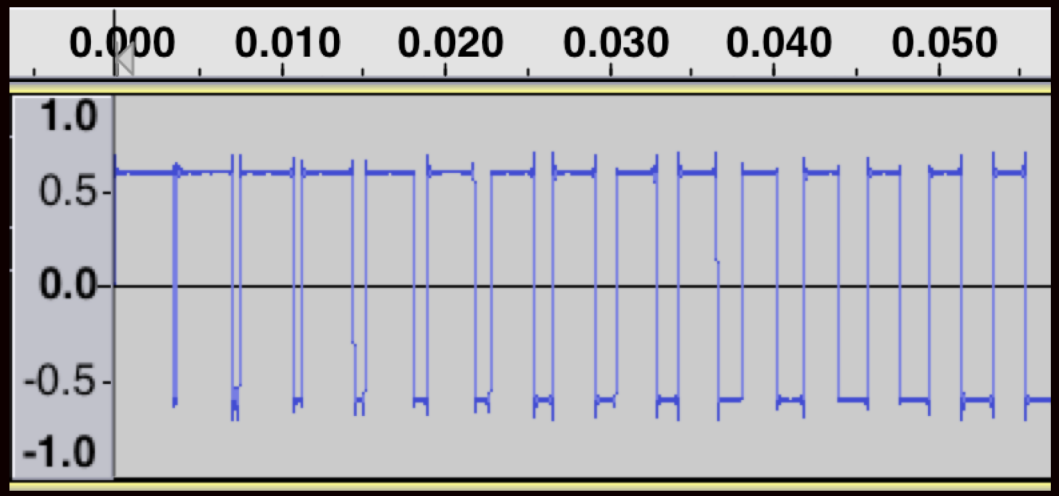


VCO2 – PWM ENVELOPE

- Envelope modulation of pulse-width using an envelope can be used to timbrally articulate a sound.

```
kpw  expseg  0.97, 0.2, 0.5, p3-0.2, 0.5
asig vco2    0.6, 220, 4, kpw
```

- In the past this has been used to imitate the articulation (attack) of brass instruments but this approach and sound is a little dated nowadays. (example 2)



squinewave

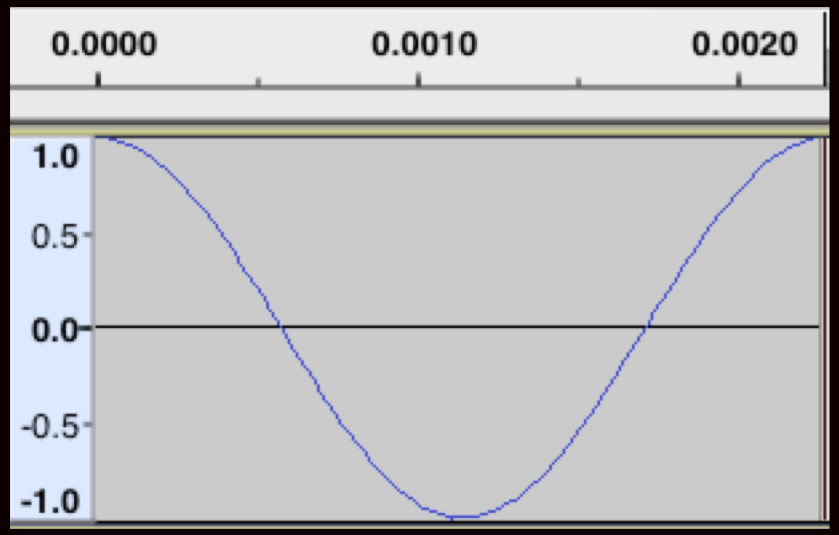
- **squinewave** creates a waveform by joining three segments: the first half of cosine, a flat portion and the second half of a cosine wave.
- By varying the proportions of these three segments we can morph between sine, sawtooth, square and pulse waveforms

```
aout [, asyncout] squinewave acps, aClip, aSkew [, asyncin] [, iMinSweep] [, iphase]
```

- The controls we will consider here will be the **aClip** argument, which controls the proportion of the flat portion, and **aSkew** which weights the proportions of the first half and second half of the cosine wave.
- Both these controls need to be a-rate.
- We will not consider squinewave's **syncout** and **syncin** options
- **iMinSweep** specifies a restriction on sudden changes in amplitude in the waveform – this can therefore be used in bandlimiting and as a soft tone control.

squinewave

```
;                                       clip   skew   syncin
asig, as   squinewave   a(icps), a(0), a(0), a(0)
```



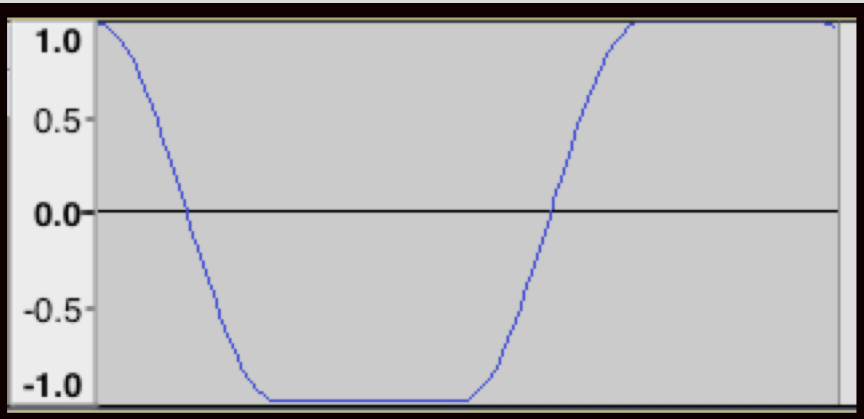
sine/cosine

- A simple sinusoidal shape (cosine).
- N.B. **asyncin** is not an optional argument, as suggested in the documentation

squinewave

- Increasing clip above zero increases the proportion of the flat portion.

```
;                                    clip     skew  syncin
asig, as  squinewave  a(icps), a(0.5), a(0), a(0)
```



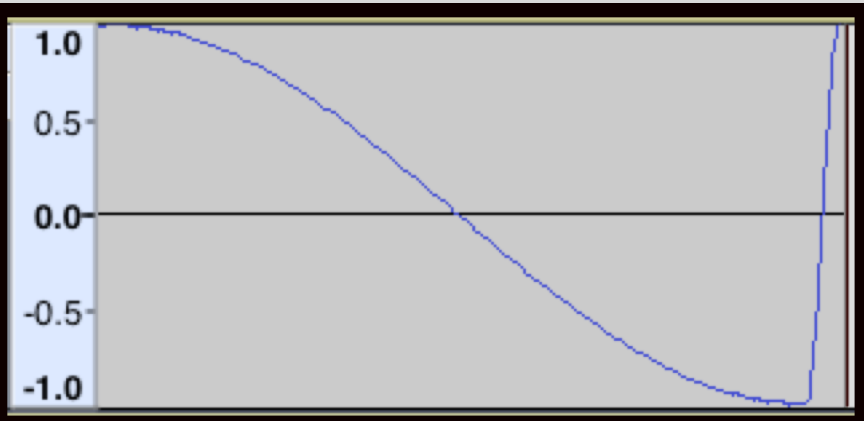

```
;                                    clip     skew  syncin minsweep
asig,as squinewave a(icps), a(0.5), a(0), a(0),  4
```

‘Square’ wave

squinewave

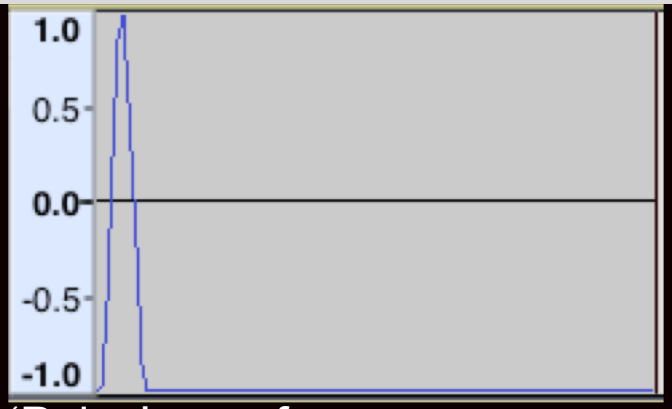
- **skew** values closer to -1 and +1 allow access to sawtooth and pulse waveforms.

```
;                                       clip   skew    syncin
asig, as   squinewave   a(icps), a(0), a(-1), a(0)
```



'Sawtooth' waveform

```
;                                       clip   skew    syncin minsweep
asig,as squinewave a(icps), a(1), a(-1), a(0),  4
```



'Pulse' waveform

GBUZZ

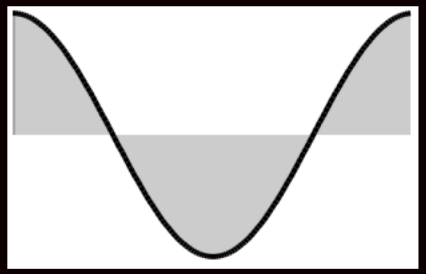
- **gbuzz** creates dynamic spectra based on stacks of harmonically related cosine waves.

```
gicos ftgen   0,0,131072,11,1    ; a cosine wave

; snip

knh   =       30                 ; number of harmonics
klh   =       1                  ; lowest harmonic
kmul  line    0, p3*0.5, 0.9, p3*0.5, 0
                                 ; power multiplier
kmul  gbuzz   kamp, icps, knh, klh, kmul, gicos
```

- gbuzz requires us to supply it with a cosine wave. We can do this with GEN 11.



Cosine Wave

GBUZZ

- **knh** (number of harmonics) defines the extent of the harmonic spectrum in harmonic partials above the lowest partial.

- **klh** (lowest harmonic) is the lowest harmonic partial to include in the stack. To start at the fundemental this should be 1.

- Although **knh** and **klh** can be changed at k-rate, they actually only change in integer steps, therefore clicks can occur when partials enter or leave.

- **kmul** controls the amplitude weighting of the partials:
  - values zero to 1 favour lower partials.
  - 1 results in all partials being equal in strength.
  - values greater than 1 favour the upper partials.

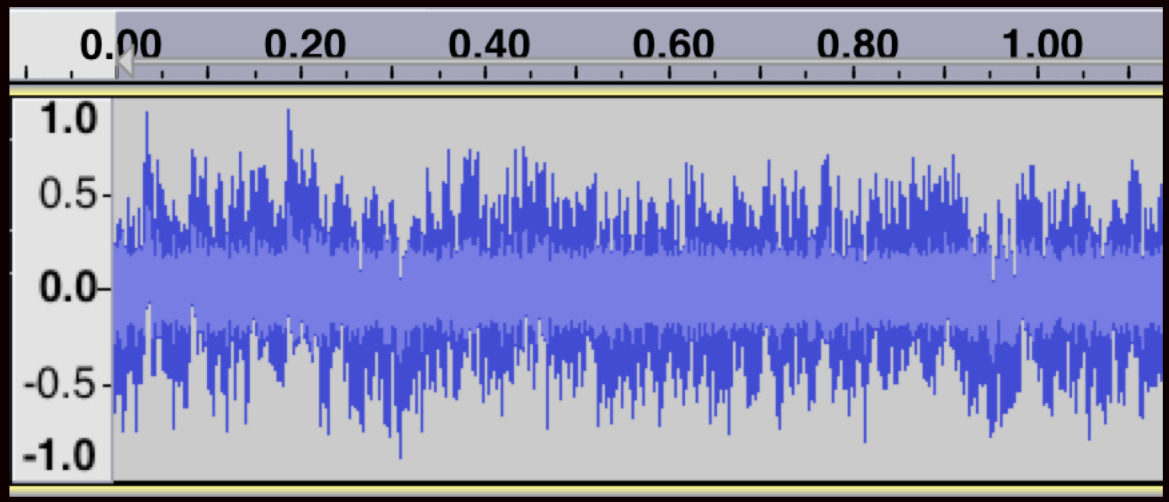
- Modulating **kmul** can be used to produce dynamic spectra.

- See example 3.

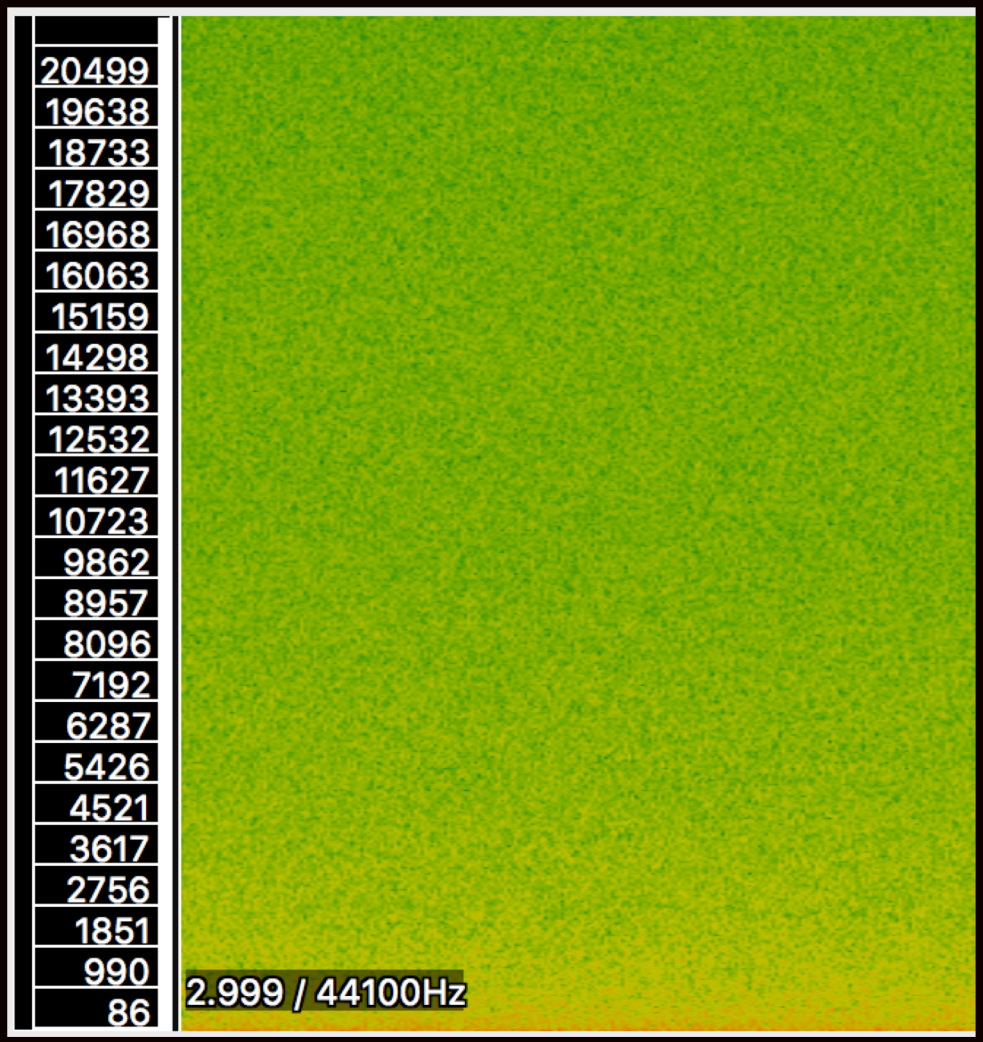
VARIETIES OF ‘NOISE‘

- ‘Noise’ can be dull or unpleasant on its own but offers possibilites for refinement through subtractive synthesis.
- ‘White’ noise presents random noise with equal energy at all frequencies. However we perceive frequency with a logarithmic scale so this in not considered ‘musical’ and can sound excessively bright and shrill.
- ‘Pink’ noise presents random noise with equal energy in all octaves. This can be considered more ‘musical’ therefore is normally more useful as a sound source in subtractive synthesis.
- ‘Dust’ can be considered noise, wherein the noise particles are interspersed with gaps of silence. Differing densities of dust can be given poetic descriptions such as ‘wind’, ‘sand’, ‘gravel’, ‘grit’ etc.

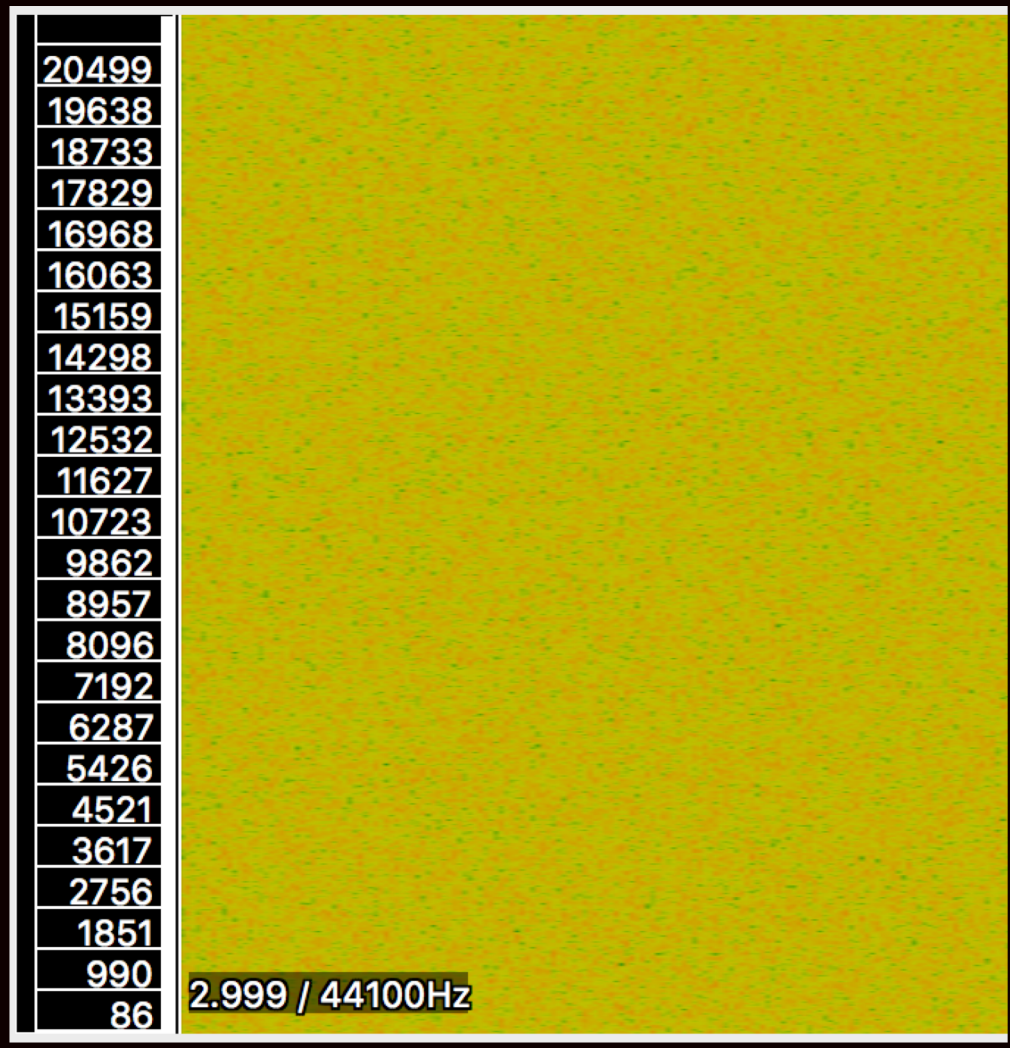
Pink Noise



VARIETIES OF ‘NOISE‘



Pink Noise



White Noise

NOISE OPCODES

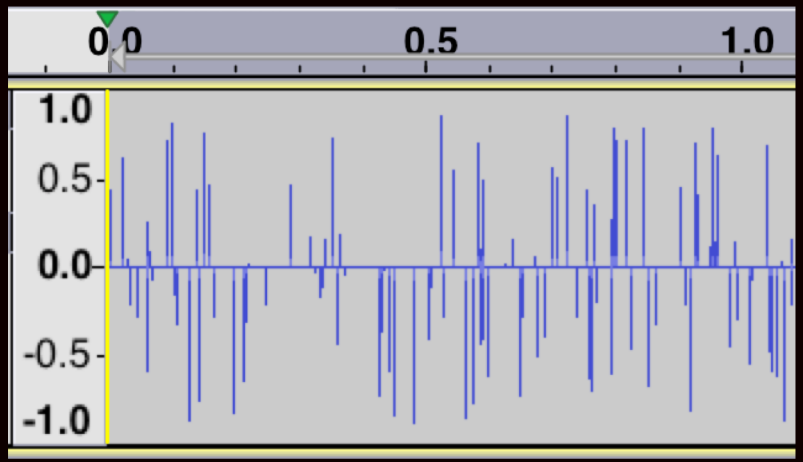
```
kBeta          =       0
aWhiteNoise    noise   kAmp, kBeta
```

- **kBeta** controls an internal lowpass filter but for the generation of white noise this should be zero.

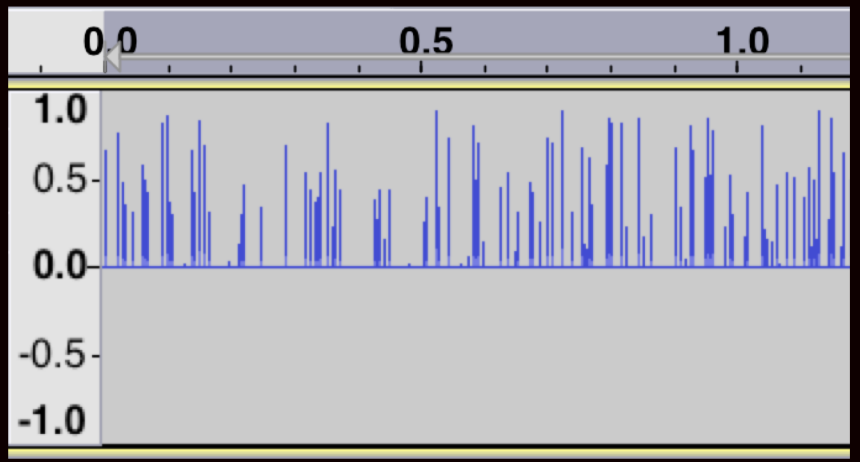
```
aPinkNoise   pinkish   kAmp
```

```
aDust   dust2   kAmp, kDensity
```

- **dust2** produces bipolar noise, dust produces unipolar noise.



Bipolar Noise (dust2)

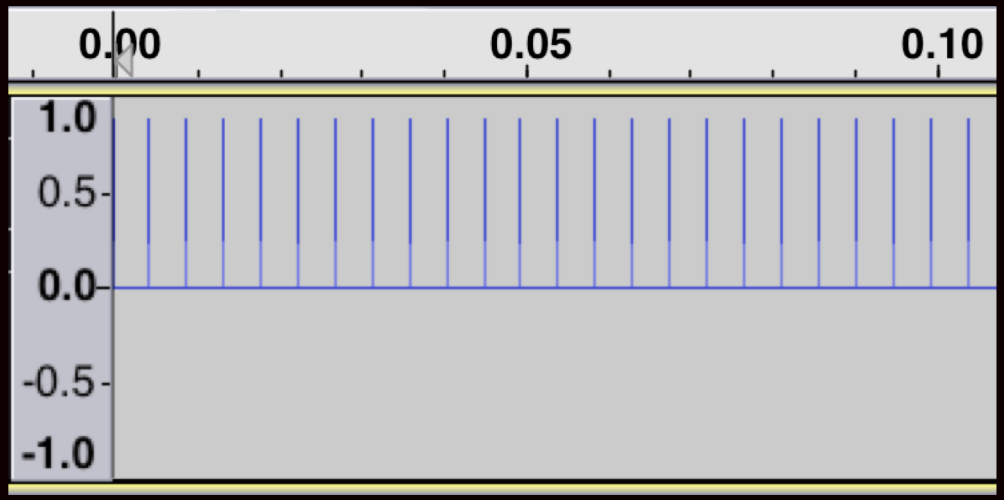


Unipolar Noise (dust)

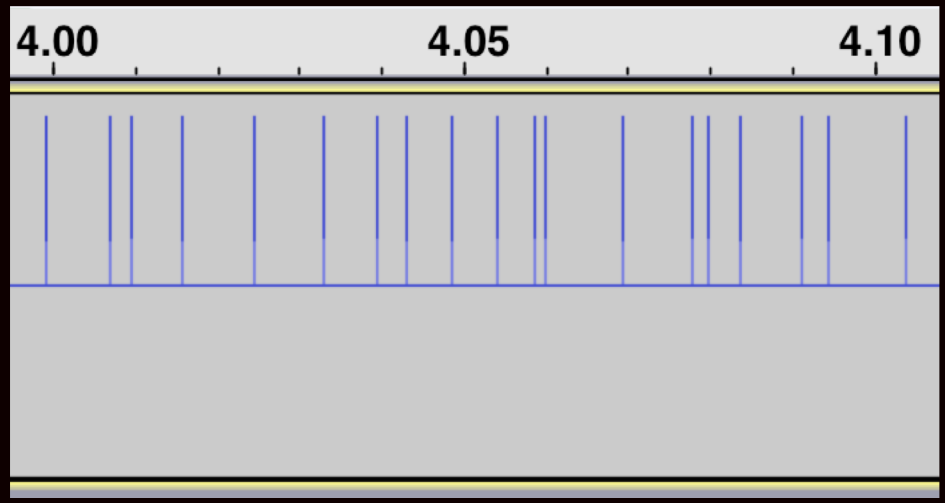
NOISE OPCODES - gausstrig

```
aSig  gausstrig  kAmp, kCPS, kDev
```

- With **gausstrig** we can move between periodic noise (a thin buzzing sound) and aperiodic noise by varying its **kDev** parameter.
- Its output is unipolar.
- Density of the noise or 'pulses' is controlled by **kCPS**.
- Pulse amplitudes could also be randomly scattered by controlling **kAmp** with a random function.



Periodic (kDev=0, kCPS=220)



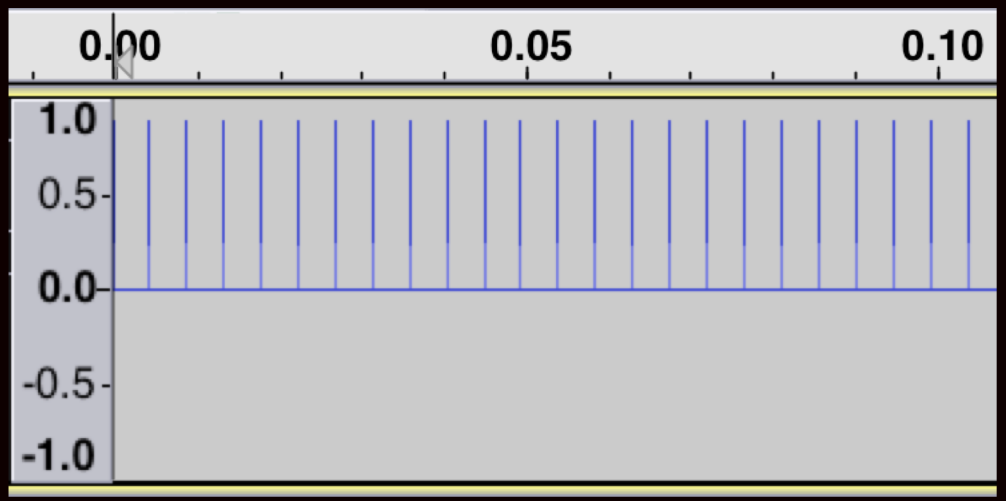
Aperiodic (kDev=1, kCPS=220)

- Example 4

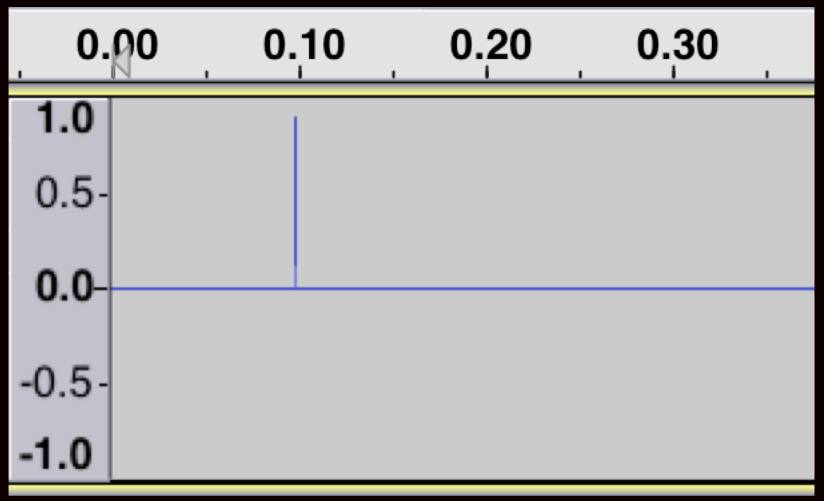
NOISE OPCODES - mpulse

```
aSig  mpulse kAmp, kGap
```

- **mpulse** creates a train of impulses similar to **gausstrig** with no deviation applied.
- Instead of specifying a frequency we specify the time gap (**kGap**) between impulses. Therefore we can derive the resulting frequency as **1/time_gap**.
- We have a special setting: if we set time gap to zero then **mpulse** will then just produce a single impulse and then nothing more. This can be useful when using subtractive synthesis for creating percussive sounds.



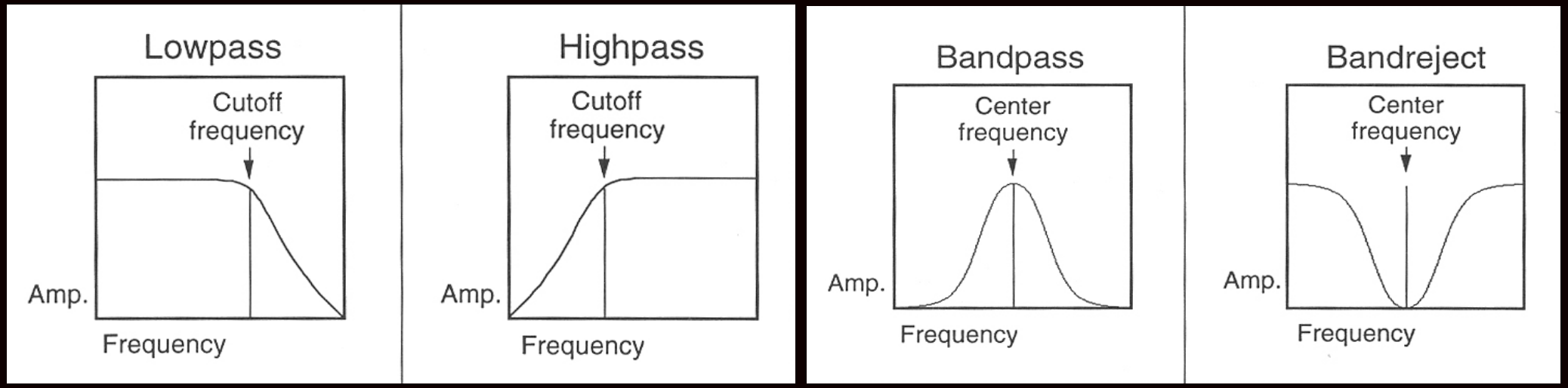
A train of impulses



A single impulse

BASIC FILTERS

- The four common filter types are:
  - Lowpass
  - Highpass
  - Bandpass
  - Bandreject (notch)
- **Lowpass** (or **highcut**) allows frequencies below its cutoff frequency to pass unattenuated.
- **Highpass** (or **lowcut**) allows frequencies above its cutoff frequency to pass unattenuated.
- **Bandpass** allows frequencies at and near its cutoff frequency to pass.
- **Bandreject** (or **notch**) stops frequencies at and around its cutoff frequency.

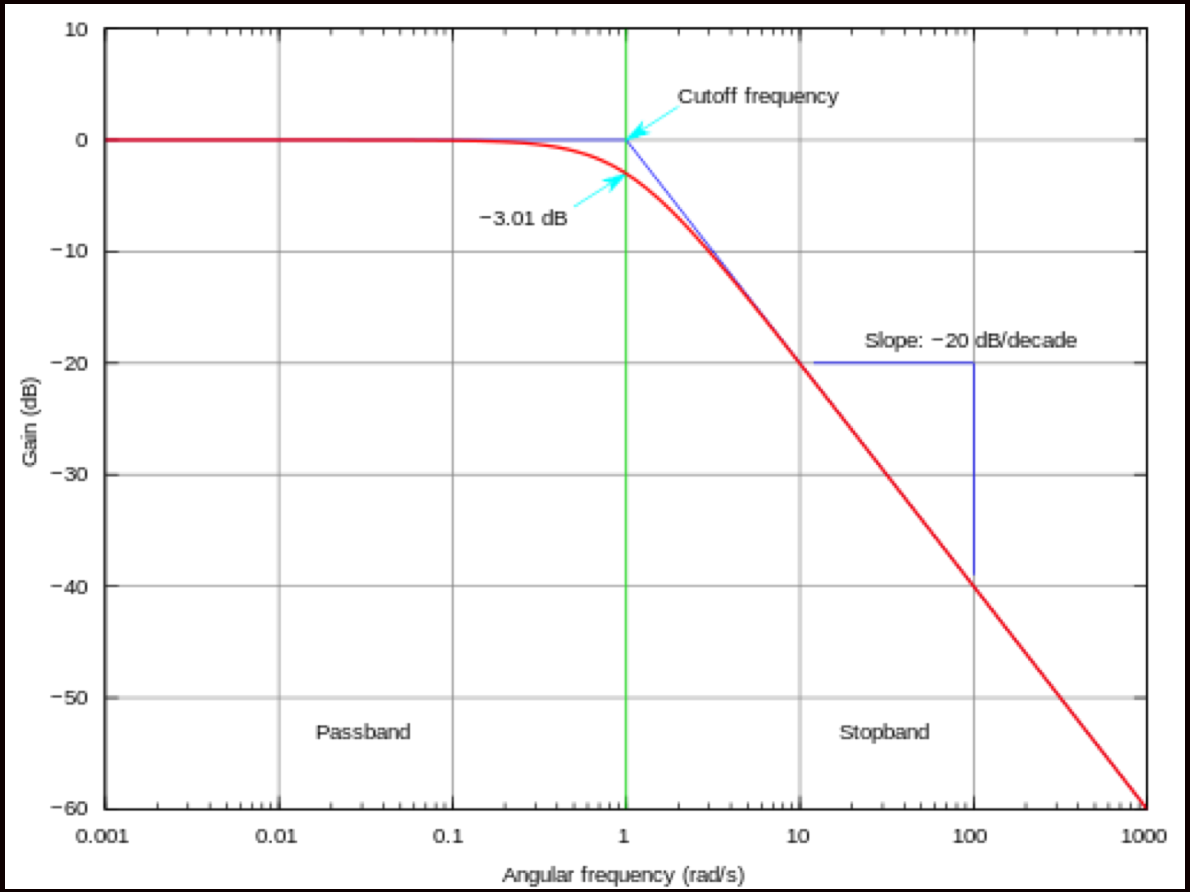


CUTOFF SLOPES

- Common cutoff slopes encountered are:
  - 6 dB/oct
  - 12 dB/oct
  - 18 dB/oct
  - 24 dB/oct

  For example: a cutoff slope of 6 dB/oct would imply that beyond the cutoff frequency, sound pressure will drop by 6 decibels for every octave increase in frequency
- Steeper slopes are not necessarily better:
  - Gentler slopes might be better for more subtle filtering
  - Steeper sloped filters can sound more dramatic and can draw attention to themselves

ANATOMY OF A CUTOFF SLOPE



- This is the response curve for a lowpass filter.
- Notice the filter has already started to remove frequencies below the cutoff point.
- The cutoff point is defined as the point at which 3dB of attenuation has occurred.

CSOUND FILTERS

- Csound possesses a vast range of filters

```
areson lowpass2 lowres lowresx lpf18 moogvcf moogladder reson
resonr resonx resony resonz rezzy statevar svfilter tbvcf vlowres
bqrez atone atonex tone tonex biquad biquada butterbp butterbr
butterhp butterlp clfilt aresonk atonek lineto port portk resonk
resonxk tlineto tonek  dcblock dcblock2 pareq rbjeq eqfil nlfilt
filter2 fofilter hilbert mode zfilter2
```

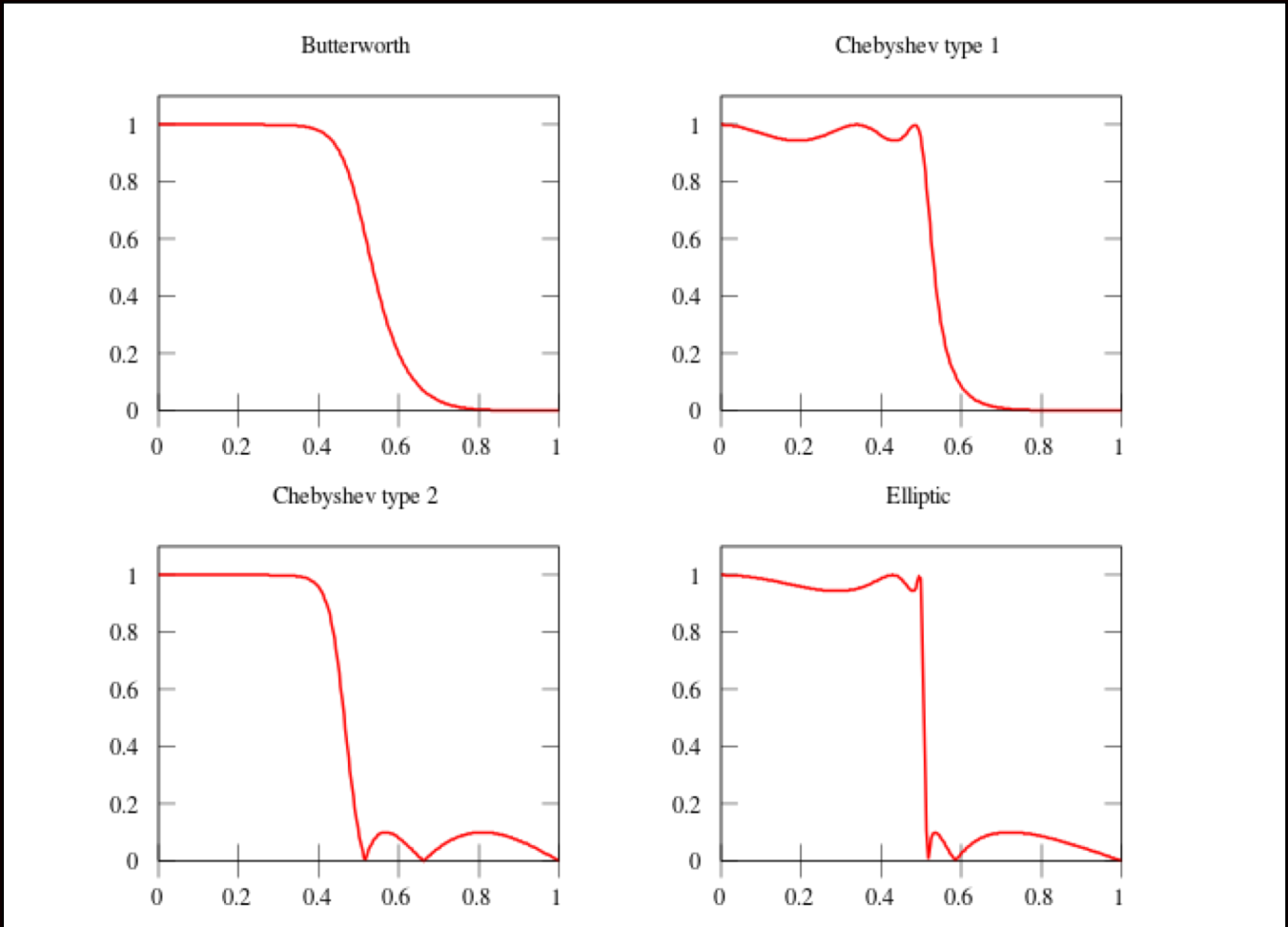
- A good place to start is the butterworth filters.

```
ares  butlp  asig, kcf          ; lowpass filter
ares  buthp  asig, kcf          ; highpass filter
ares  butbp  asig, kcf, kbw     ; bandpass filter
ares  butbr  asig, kcf, kbw     ; bandreject filter
```

- **kcf** = cutoff frequency
- **kbw** = bandwidth

SPECIFIC FILTER MODELS

- Different filters built upon different mathematical models (which often exist in both the analogue and digital domains) exhibit different characteristics. None are ideal.



- Butterworth offers a very flat passband so is often favoured in muscial applications.

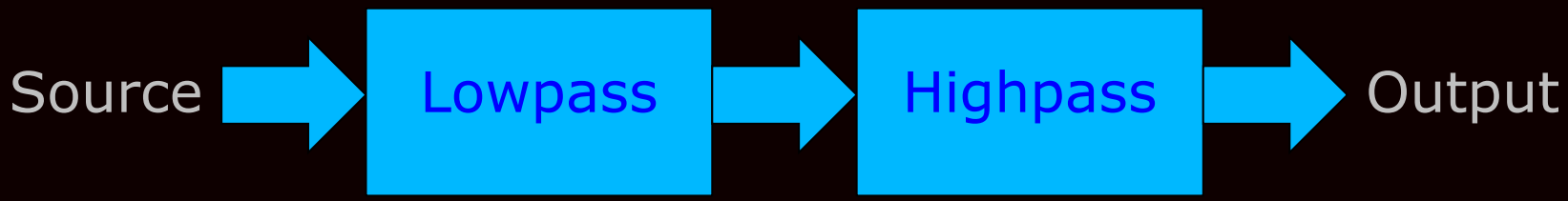
FILTERING EXAMPLE

- Lowpass filtering a sawtooth waveform

```
aSig   vco2   0.4, 220       ; create a sawtooth
kCF    expon  2000, p3, 50   ; create a filter cutoff envelope
aSig   butlp  aSig, kCF      ; filter the sawtooth
       outs   aSig, aSig     ; send to the speakers
```

- **expon** is a variant of the **line** opcode in which the connecting segment is an exponential curve rather than a straight line. (Values should be alike in sign and the envelope should not cross zero.)
- It is acceptable to overwrite the input of an opcode with its output if we have no further use of the version provided by the input. This technique can simplify coding by leaving us with fewer variables to keep track of. It also saves RAM but this saving is fairly negligible on modern computers.
- **kCF** can also be a-rate; this might be a useful option if the cutoff will be changing at a high rate.

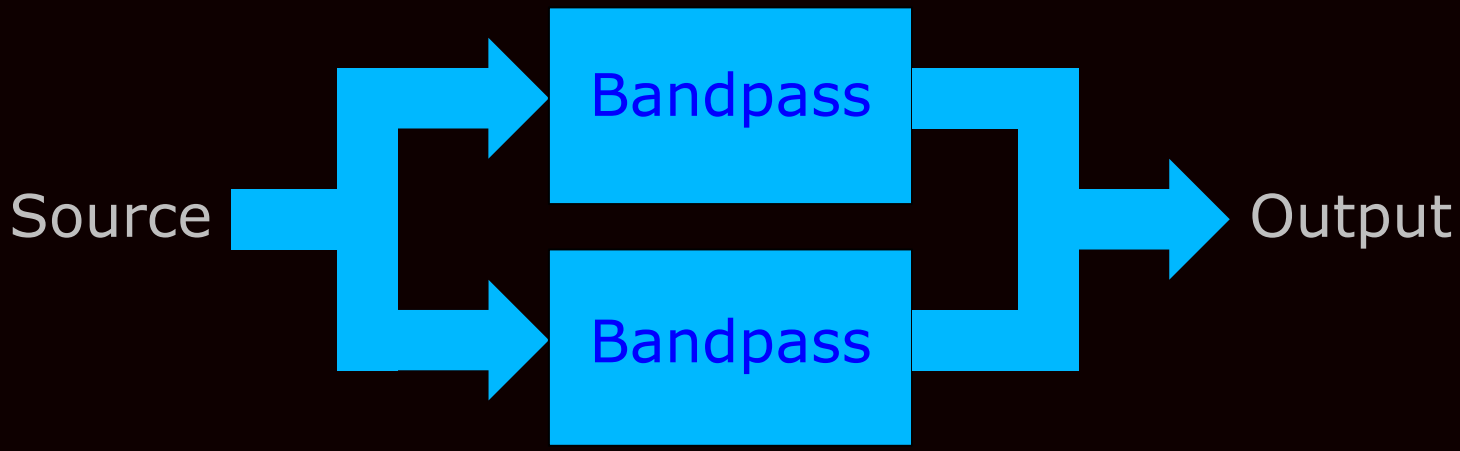
SERIES CONNECTION




```
aSig   vco2    0.4, 200, 2, 0.5 ; square wave
aSig   butlp   aSig, 5000       ; lowpass filter
aSig   buthp   aSig, 2000       ; highpass filter
       outs    aSig, aSig       ; output
```

- Order of the filters doesn’t matter.
- If the cutoff frequency of the highpass filter is above that of the lowpass filter then the signal can disappear completely.
- This arrangement can be used as a kind-of bandpass filter.

PARALLEL CONNECTION




```
aSig   vco2    0.4, 111, 2, 0.5 ; square wave
aBP1   butbp   aSig, 1000, 333  ; bandpass filter 1
aBP2   butbp   aSig, 2000, 667  ; bandpass filter 2
aMix   =       aBP1 + aBP2      ; mix the two filters
       outs    aMix, aMix       ; output
```

- Order of the filters doesn’t matter.
- Note that we create two new audio variables as the outputs of the two bandpass filters.
- It is suggested to create another audio variable for the mix rather than mixing them at the output.

STATIC FILTERS vs FUNDAMENTAL-FOLLOWING FILTERS

- It is equally valid to design a subtractive synth within which a filter cutoff remains fixed or one which moves in ratio with changes in the fundamental frequency of the sound source.
- The reasons for choosing one or the other relate to the physics you might be emulating:
    - Filtering that is physically inate to the exciter (such as a guitar string) will shift in ratio with the source sound.
    - Filtering that is physically inate to a resonator (such as a guitar body) will remain fixed.
- See examples 5 and 6.

FUNDAMENTAL FOLLOWING FILTERS

- Also commonly referred to as 'key following'.

```
iCPS    cpsmidi                          ; read in MIDI notes
aSig    vco2    0.4, iCPS                ; create a sawtooth
iBW     =       0.7 * iCPS               ; bandwidth
iRatio  =       5                        ; C.F ratio
aSig    butbp   aSig, iCPS*iratio, iBW   ; filter the sawtooth
        outs    aSig, aSig               ; send to the speakers
```

- In this example the bandpass filter will always accent the 5th harmonic partial, regardless of what note is played.
- It is also common practice to express the bandwidth of the bandpass filter in ratio with its frequency. This can be considered more 'musical' as the bandwidth control (in this example 0.7) will be expressing a fraction of an octave rather than a fixed number of cycles-per-second.
- It is also possible to devise a filter that is partially static and partially key-following by using a cutoff value that is an interpolation between a static and a key following value.