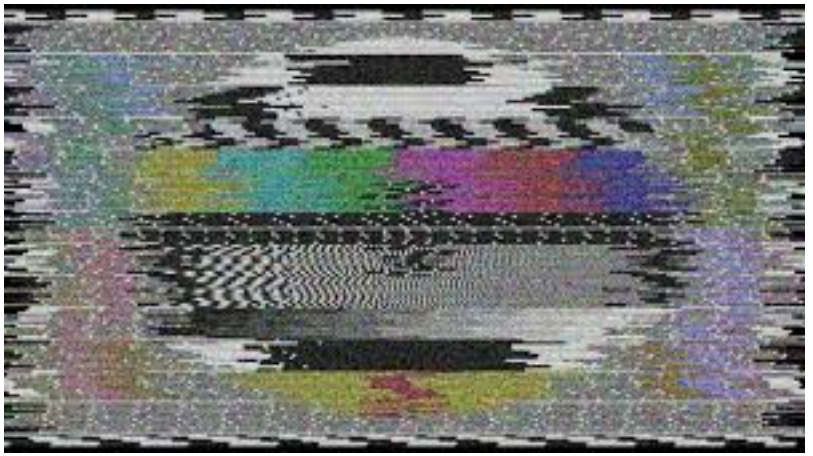
GLITCH TECHNIQUES

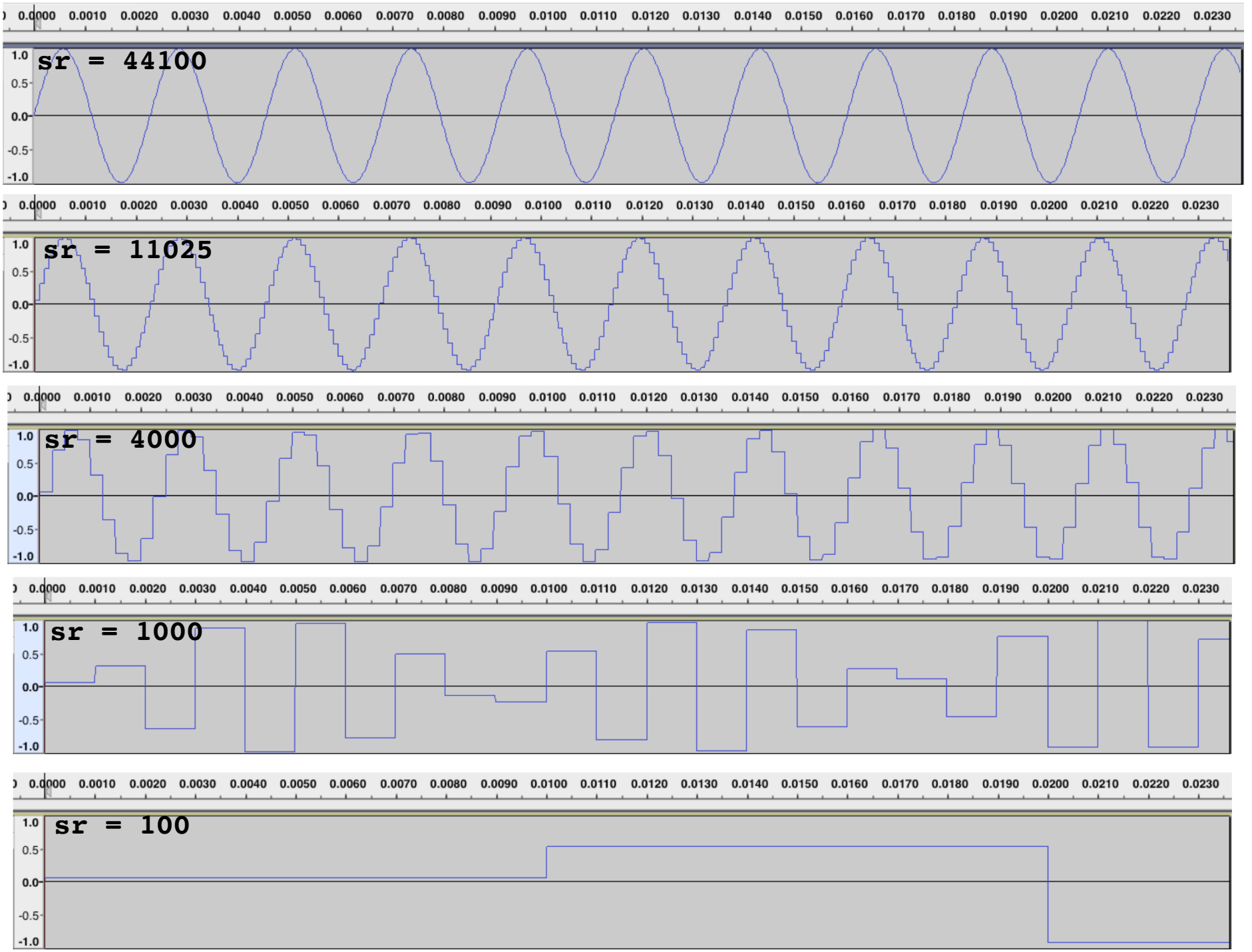
DEFINITION

- As an aesthetic reaction to the increasingly fidelity of digital audio and complexity and richness of synthesised sound, interest has developed in exploring lower fidelity and sonic artefacts that would normally be shunned.
- The ‘glitch’ aesthetic largely emerged in the 1990s and is exemplified in the work of people like Ryoji Ikeda, Autechre and Aphex Twin.
- This lecture will look at at implementations of a number of glitch, noise and distortion techniques within Csound.



SAMPLE RATE REDUCTION - DECIMATION

- Reducing the sample rate will lower the Nyquist frequency (**`sr/2`**) – the highest frequency reproducible.
- At sample rates such as 44100 Hz these frequencies are beyond the range of our hearing, but at lower sample rates, frequencies that would normally be audible but are now above the new Nyquist frequency become distorted producing quantisation artefacts.
- If we change the sample rate on our sound card (or via Csound **`sr = ...`**) then our sound card hardware will apply filters to reduce these artefacts.
- We can however ‘force’ artefacts by replicating a reduction in sample rate within our instrument code.
- The next slide illustrate a 440 Hz sine tone reproduced at a range of sampling rates.


sr = 44100
sr = 11025
sr = 4000
sr = 1000
sr = 100

SAMPLE RATE REDUCTION - DECIMATION

- The fold opcode makes sample rate decimation easy.

```
aResult   fold   aInSig, kDiv
```

- **kDiv** represents a factor by which the current Csound sampling rate will be divided, therefore if **kDiv** is 1 no change will be applied.
- If **kDiv** is 2 and **sr = 44100** the result will be as if sampling rate was 22050.
- The code below will implement a desired sample rate, in this case 8000 Hz.

```
kNewSR    =       8000
kDiv      =       sr / kNewSR
aResult   fold    aInSig, kDiv
```

BIT DEPTH REDUCTION – BIT CRUSHING

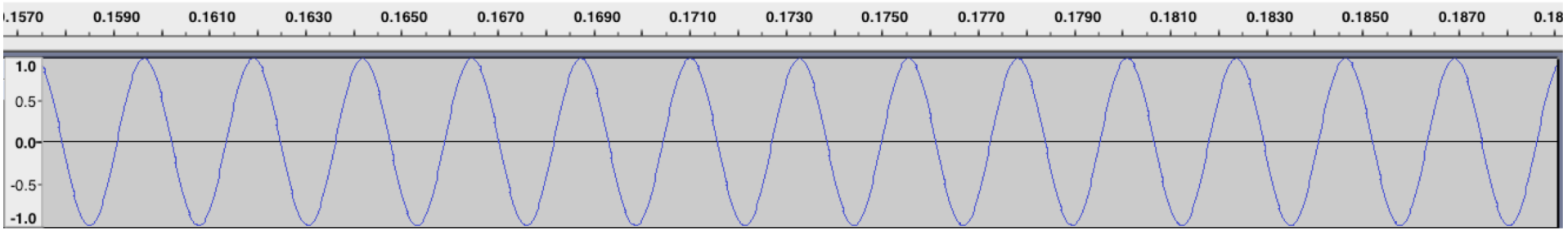
- 'Bit depth' refers to the number of discrete amplitude values that can be stored in digital audio.
- Bit depth essentially defines the dynamic range of digital audio. If bit depth is drastically reduced, hiss and crackling and a kind of gating effect will result.
- Using the **`round()`** function to integerise amplitudes along with some other mathematics, can produce the quantisation associated with bit depth reduction.
- For example the code below will emulate 8 bit audio

```
iBits  =  8
kVals  =  (2 ^ iBits)
kAmp   =  kVals / 2
aSig   =  round(aSig * kAmp) / kAmp
```

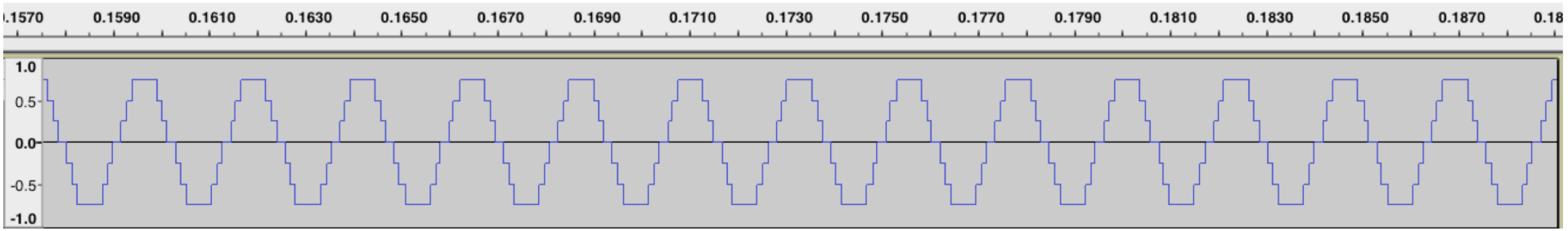
BIT DEPTH REDUCTION – BIT CRUSHING

- A 440 Hz sine wave at different bit depths.

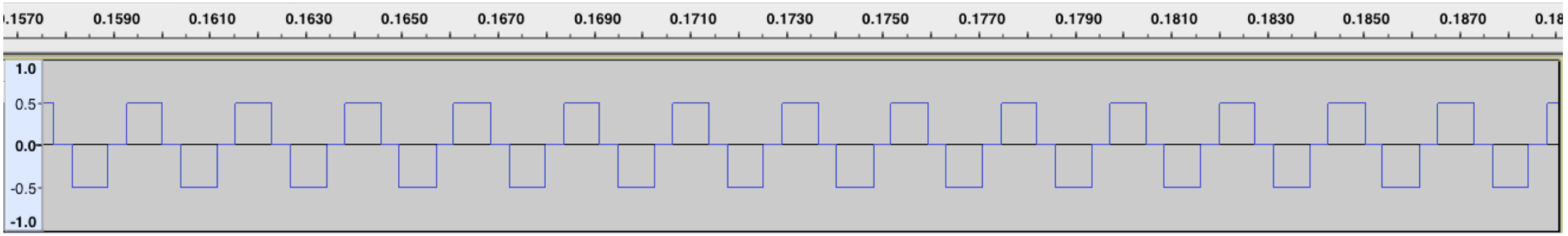
bit depth = 16
2 ^ 16 (65536 values)



bit depth = 3
8 values



bit depth = 2
4 values



CLIPPING

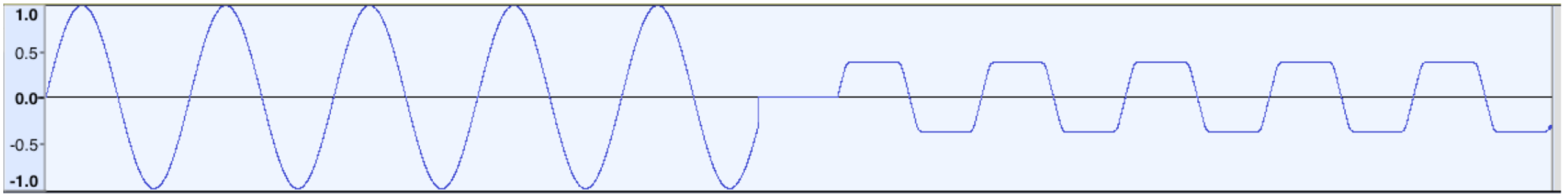
- 'Clipping', 'distortion', more comprehensively described as 'waveshaping', is a huge area of signal processing but a simple way to produce soft clipping is with the **`clip`** opcode.

```
aResult  clip  aInSig, iMethod, iLimit
```

- **`iMethod`** refers to a clipping method, method 0 (Bram de Jong method) is recommended.

Unclipped sine wave.
Amplitude =1

Sine wave amplitude = 1.
Clipped at **`iLimit = 0.5`**

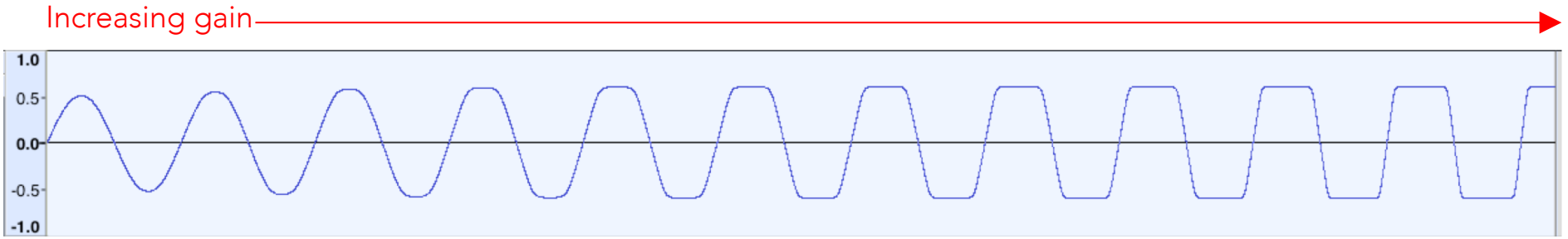


- **`iLimit`** is the absolute amplitude limit allowed to leave the opcode. Soft clipping (shaping) of the input signal will occur below this limit.

CLIPPING

- We can not change **iLimit** at **k**-rate but we can amplify the input signal which will then function as a gain control. E.g:

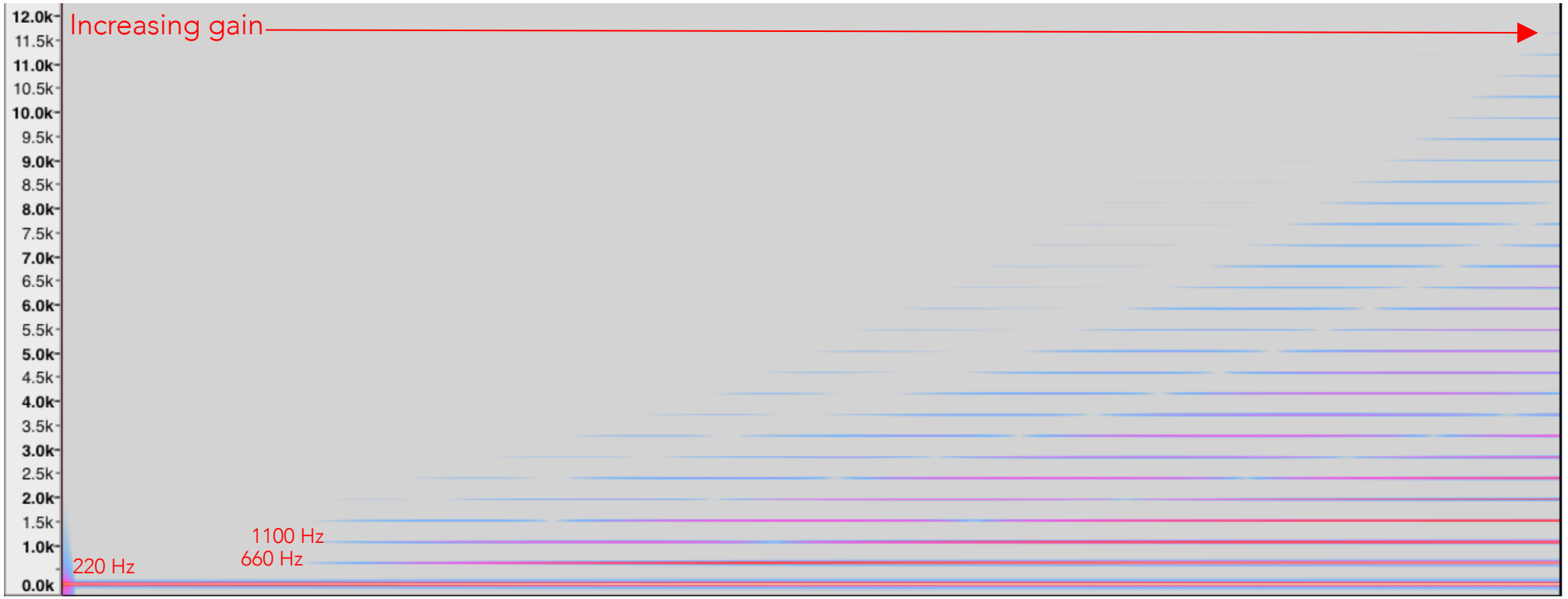
```
aSig      poscil 0.5, 110
kGain     expon  1, p3, 4
aResult   clip   aInSig * kGain, 0, 0.8
```



- Again note how the output amplitude cannot go above the limit set (in this case 0.8).

CLIPPING

- Spectrally, this sort of clipping will add odd numbered harmonics. In the spectrogram below, an increasing amount of clipping is applied to a 220 Hz sine tone.

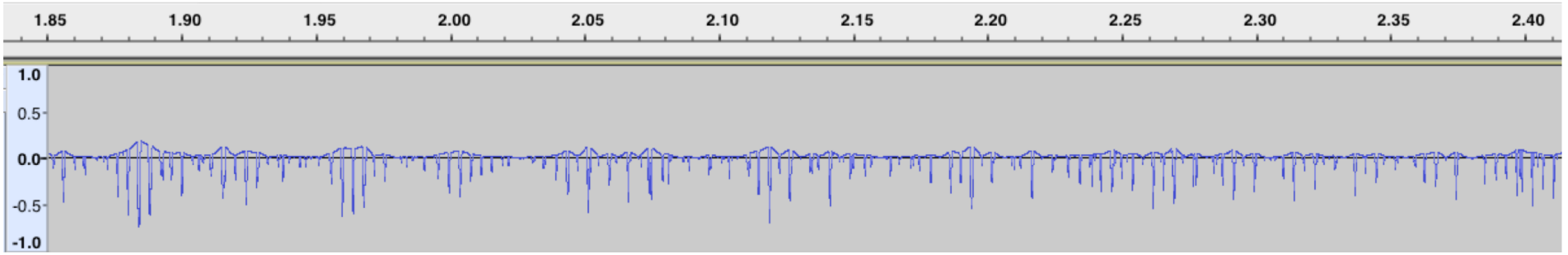


- To control the tone produced by clipping, consider the effect of lowpass filtering both before and after the clipping procedure.

BIASED CLIPPING

- Asymmetric clipping is a phenomenon that is found in a range of functioning and malfunctioning electronics. An audio signal is clipped differently in one polar direction than the other.
- We can replicate this effect by adding a DC offset before a signal is clipped.
- We can remove this offset after clipping using DC blocking (**`dcblock2`**).

```
...
kGain       =           1.3
kBias       =           0.6
aSig        clip        (aSig + kBias) * kGain , 0, 0.8
aSig        dcblock2    aSig
...
```



CLICKS

- A click is essential a very short burst of wideband noise.
- There are a wide range of ways in which this can be synthesised.
- Most obvious options are the **dust2** and the **mpulse** opcodes.

```
aClick  mpulse  1, 2
```

- The above code will create a click of amplitude 1, once every 2 seconds.

```
aClick  mpulse  1, 0
```

- The above code will create just a single click.

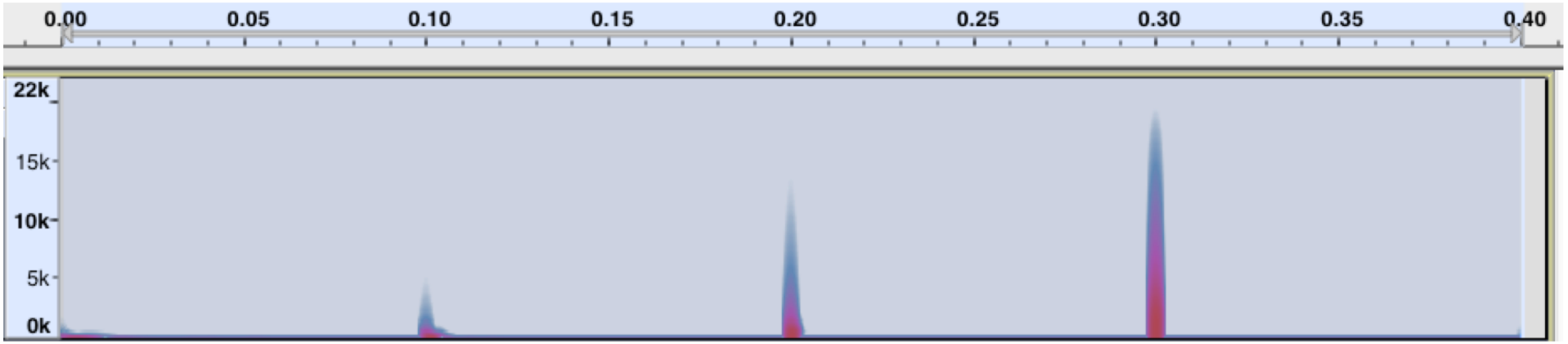
CLICKS AND FILTERING

- Just like subtractive synthesis, filtering clicks can alter their character dramatically. E.g:

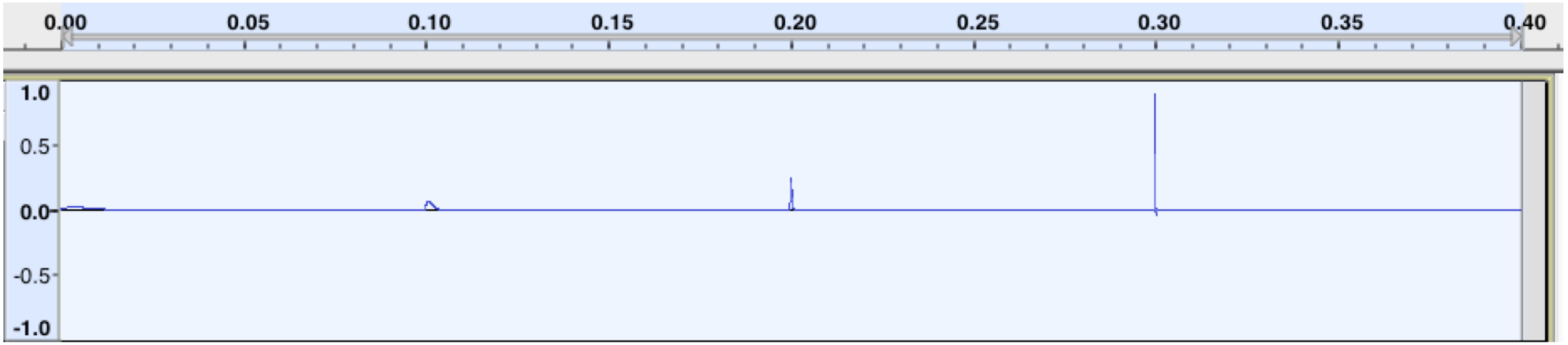
```
p3    =        0.4
aImp  mpulse   1, 0.1
aCF   expon    50, p3, 10000 ; rising filter cutoff
aImp  butlp    aImp, aCF
```

- Four clicks will be produced, each one filtered progressively less.
- The resulting waveform and spectrograms are shown on the next slide.

CLICKS AND FILTERING



spectrogram (rising lowpass filter cutoff is evident)



waveform

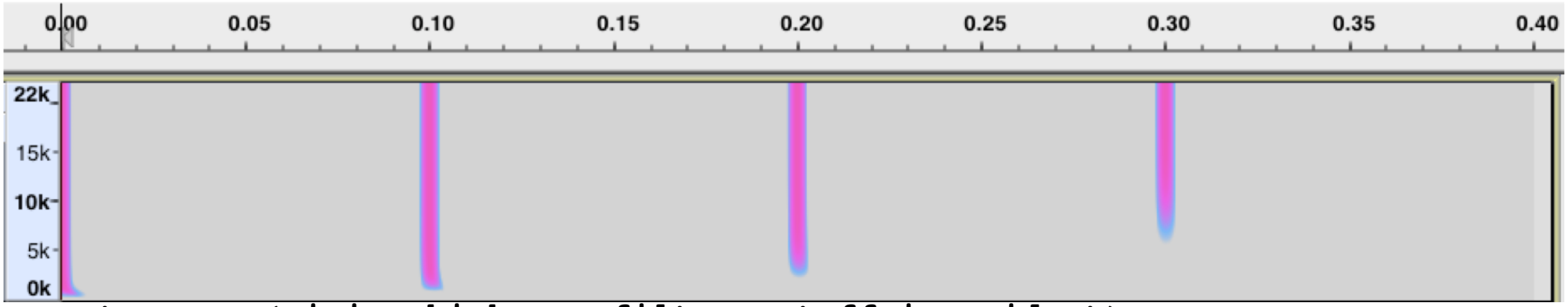
- You can notice also how the filter extends the persistence of the click. I.e. the filter ‘resonates’.

CLICKS AND HIGHPASS FILTERING

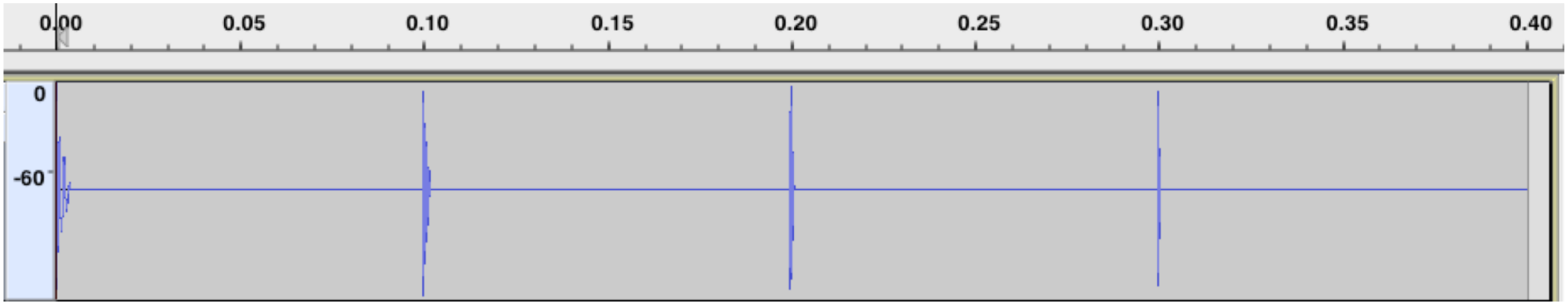
```
p3    =        0.4
aImp  mpulse   1, 0.1
aCF   expon    500, p3, 18000 ; rising filter cutoff
aImp  butlp    aImp, aCF
aImp  butlp    aImp, aCF
aImp  butlp    aImp, aCF
aImp  butlp    aImp, aCF
```

- You can pass the audio signal through a filter multiple times to steepen the cutoff slope if it seems otherwise insufficient.
- Spectrogram and waveform are shown on the next slide.

CLICKS AND HIGHPASS FILTERING



spectrogram (rising highpass filter cutoff is evident)



waveform

- Again temporal smearing as a result of the filtering process is evident.

CLICKS AND BANDPASS FILTERING

- If we bandpass filter a click with a very narrow bandwidth, the filter will start to resonate at the cutoff frequency.
- In the case of of filter like **butbp** we will lose a lot of the power of the signal so we may need to compensate for this in some way.

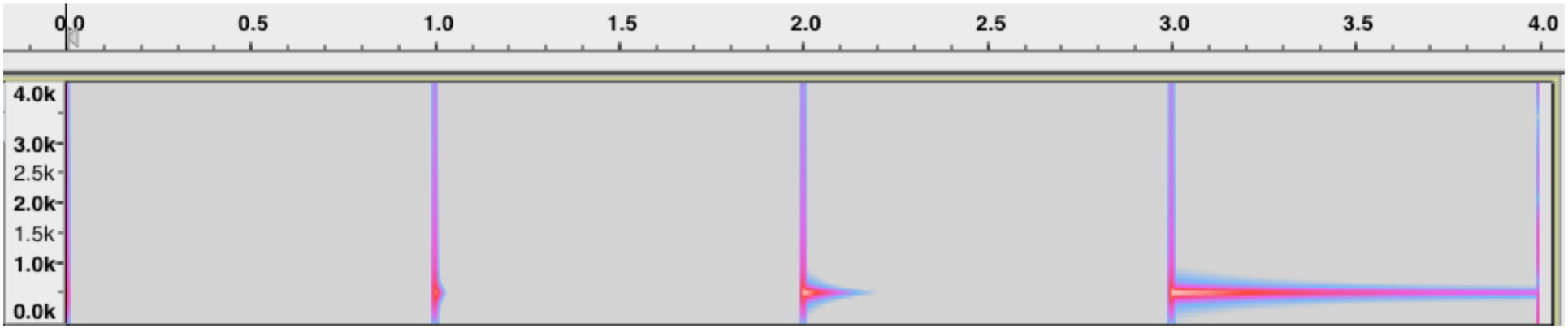
```
aBW     expon   1, p3, 0.001 ; narrowing bandwidth
aImp    butbp   aImp, 500, 500 * aBW
aGain   expon   1, p3, 700 ; gain envelope
aImp    *=      aGain     ; compensate for gain loss
```

- **reson** (also a bandpass filter) includes mechanisms for gain compensation accessed using its 4rd input parameter.

```
aImp    reson   aImp, 500, 500 * aBW, 1
```

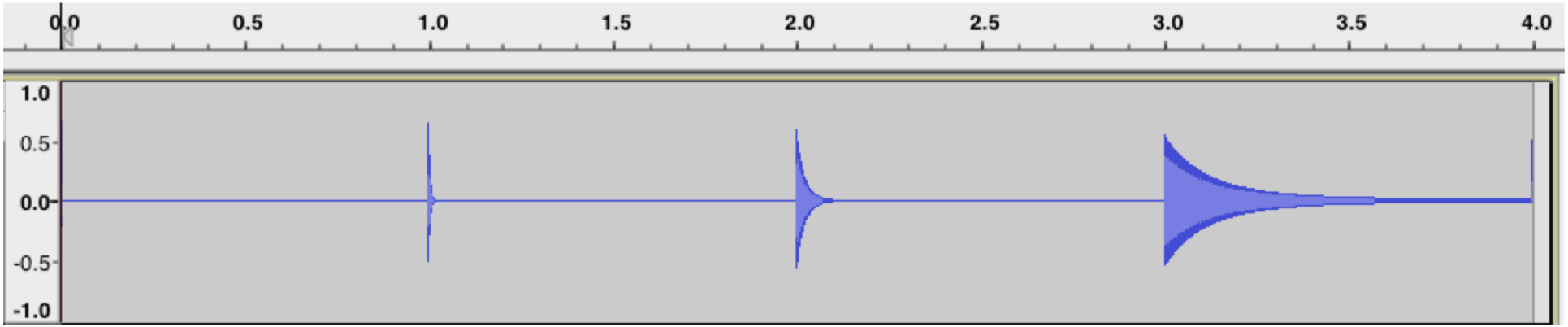
- Waveform and spectrogram on the next slide.

CLICKS AND BANDPASS FILTERING



spectrogram

- As the bandwidth narrows, focus upon the cutoff frequency increases.



waveform

- The sustain of the result sound also increases as bandwidth narrows.

VINYL CRACKLING

- Crackling can be approached as varying densities of clicks. **`dust2`** can offer a starting point.
- **`dust2`** has randomly varying amplitude built in. To augment this we can add randomly varying filtering.

```
iDensity  =         50
aDust     dust2     0.1, iDensity
kCF       randomh   5, 13, iDensity
aDust     butlp     aDust, cpsoct(kCF)
kPan      randomh   0, 1, iDensity
aL, aR    pan2      aDust, a(kPan)
          outs      aL, aR
```

- Reminder: **`randomh`** creates a random sample-and-hold function. Here it functions at a rate the same as the density of the dust ensuring a new filter position for each click.

VINYL HISS

- A similar approach but increasing density and replacing the lowpass filter with a highpass filter will produce vinyl hiss.

```
iDensity  =         5000
aDust     dust2     0.1, iDensity
kCF       randomh   5, 13, iDensity
aDust     buthp     aDust, cpsoct(kCF)
kPan      randomh   0, 1, iDensity
aL, aR    pan2      aDust, a(kPan)
          outs      aL, aR
```

VINYL POPS

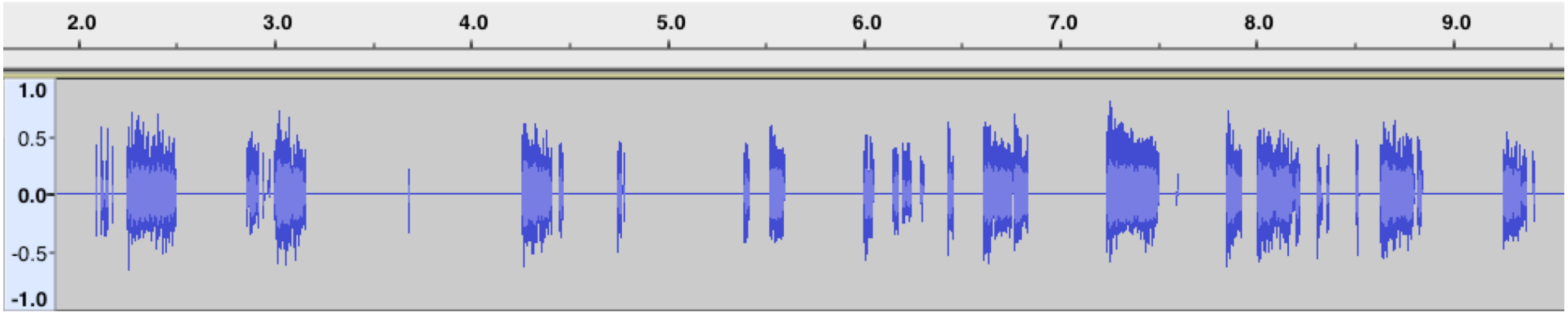
- Low density dust through a bandpass filter will produce pops with a brief but perceiveable pitch.

```
iDensity  =         100
aDust     dust2     0.1, iDensity
kCF       randomh   5, 13, iDensity
aDust     butbp     aDust, cpsoct(kCF) , cpsoct(kCF)*0.2
kPan      randomh   0, 1, iDensity
aL, aR    pan2      aDust, a(kPan)
          outs      aL, aR
```

- Take care when modulating the cutoff frequency of a bandpass filter very quickly as this will have a tendency to create accentuated clicks and pops and potentially massive overloading.

INTERMITTENCY

- An intermittent signal connection can be enforced by applying a binary gate (either on or off).



- Options can be explored with regard to how that gate is triggered.

INTERMITTENCY

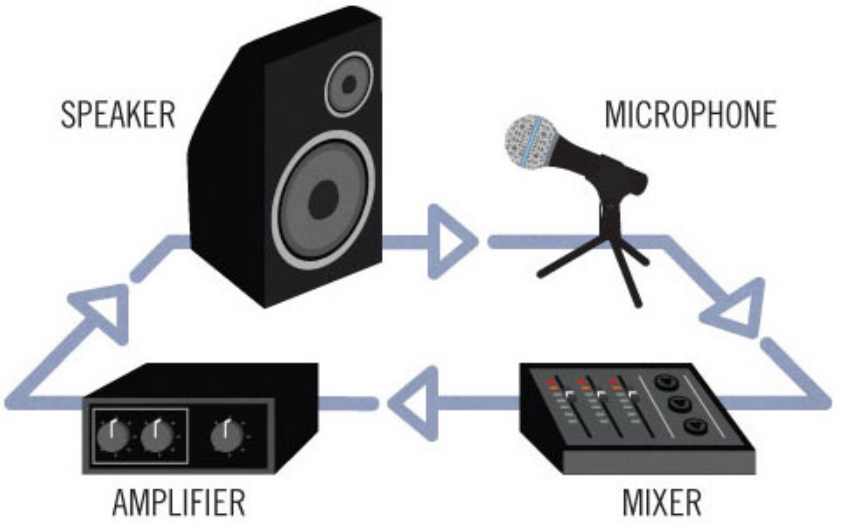
- In the real world, an intermittent signal often gates open if the input signal rises above a certain threshold. We can emulate this by tracking the RMS of the input signal and using that to define the gate status.

```
aSigL,aSigR diskin2 "ClassicalGuitar.wav", 1, 0, 1
kRMS        rms     aSigL
kThreshold  invalue "Threshold"
kGate       =       kRMS > kThreshold ? 1 : 0
            outs    aSigL*a(kGate), aSigR*a(kGate)
```

- "If **kRMS** is greater than **kThreshold**, then **kGate** is equal to 1, otherwise **kGate** equals zero".

FEEDBACK

- In the same way that sound from an speaker can be fed back into the source from which it originated: a guitar pickup, a microphone, a mixer channel.



- We can feed the audio from an sound processing opcode back into its input.
- Normally this would be expected to produce a disastrous build up of the sound but we can modify the sound further before being fed back using attenuation, a filter, a delay, clipping…

FEEDBACK

- Imagine sound being passed through an **streson** filter and including the output signal along with the input.

```
aDustSig dust2    1, 10
aStrSig  streson aDustSig + aStrSig, 220, 0.9
```

- The output signal from **streson** '**aStrSig**' is fed back into the input.
- But there is a problem in that the first time that Csound evaluates the code, **aStrSig** does not exist.
- The way around this is to initialise the variable using init; this assigns it a value at **i**-time (it will be overwritten by **streson** thereafter).

```
aStrSig  init  0
...
```

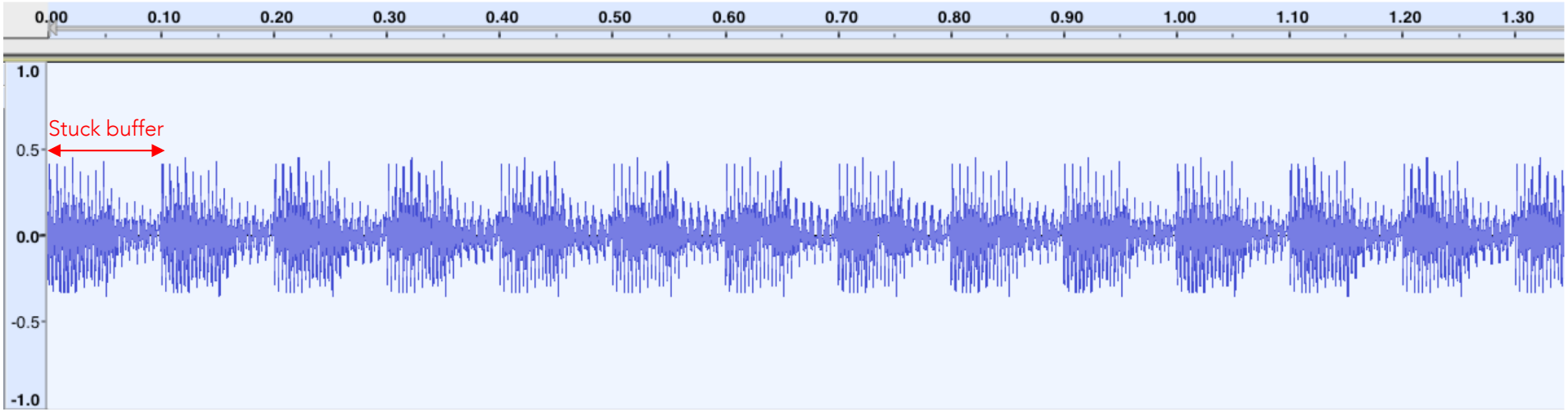
FEEDBACK

```
...
aSig  init     0
aSig  streson  aImp + (aSig * 0.2), 50, 0.9
kCF   rspline  4, 11, 0.4, 0.8
aSig  butbp    aSig, cpsoct(kCF), cpsoct(kCF)*5
aDel  rspline  0.01, 0.03, 0.01, 0.02
aSig  flanger  aSig, aDel, 0, 1
aSig  clip     aSig, 0, 0.5
...
```

- A randomly modulating bandpass filter and delay ensure variability in the sound.
- Clipping the signal (**`clip`**) ensures that things can't get too unruly.
- The feedback ratio is 0.2 (20%).

STUCK BUFFER

- Computers process audio signals in chunks called buffers.
- The duration of a buffer might range from 0.01 to 0.5 seconds.
- If a computer crashes sometimes the program continue to repeat the last received buffer in a loop.



- In the audio shown above, a buffer of size 0.1 seconds has become stuck.

STUCK BUFFER

- We can imitate a stuck buffer using a carefully time envelope on the input sound in conjunction with a delay with 100% feedback.

```
...
iBufLen  =        0.1
aEnv     linseg   1, iBufLen, 1, 0, 0
aSig     =        aSig * aEnv
aStuck   flanger  aSig, a(iBufLen), 1
         out      aSig + aStuck
```

- Note how the input sound, **aSig**, is gated by the envelope **aEnv** and immediately it is gated the output from the delay (**flanger**) will take over.
- (a more flexible method is shown in the .csd example)
- The input signal **aSig** could be from a sound file or from the live input.