noise.rb

How to annoy your cat with sound generators

Arlington Ruby, May 2012

Brock Wilcox
awwaiid@thelackthereof.org

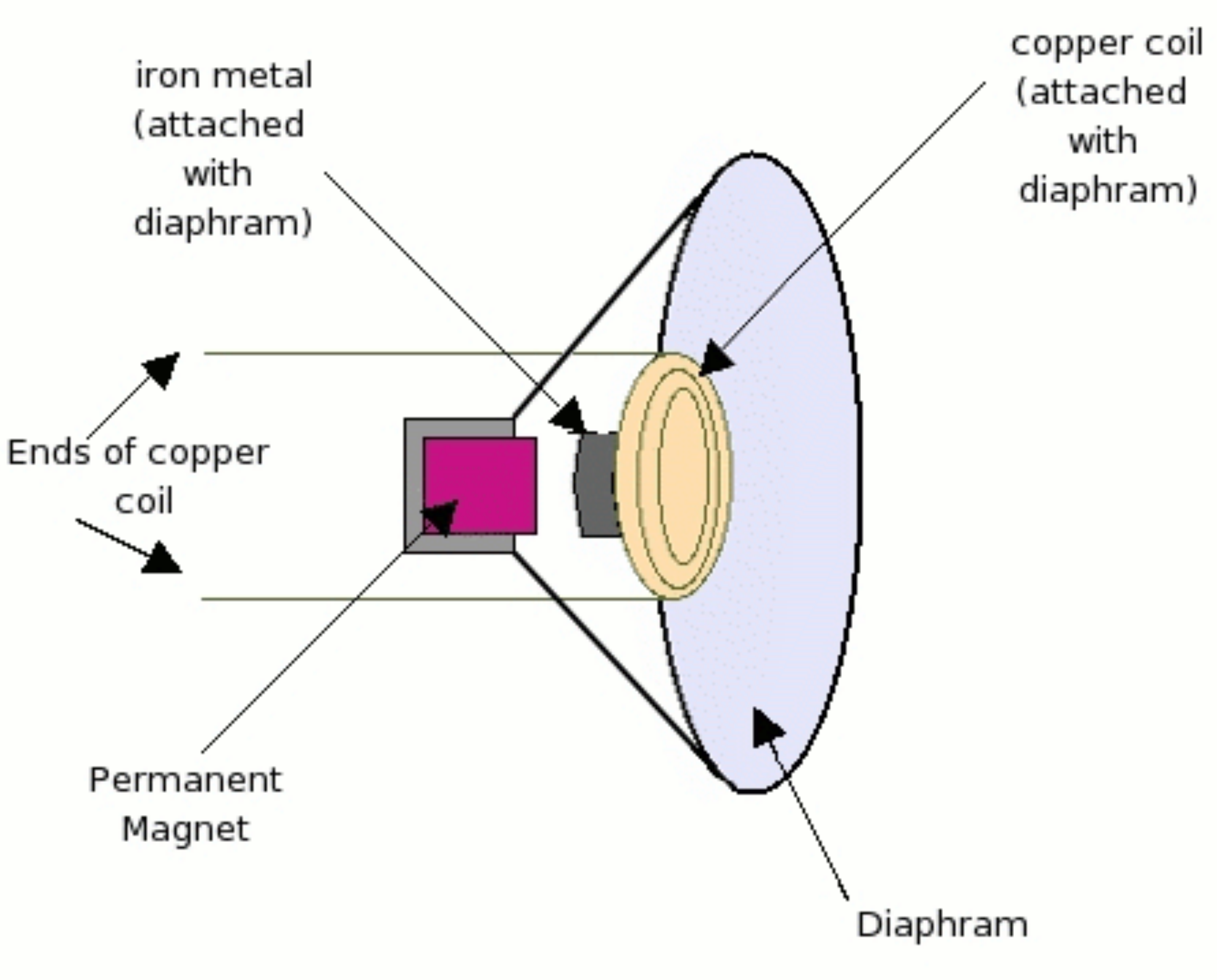
Let's build a software synthesizer in about ~~20~~ 40 minutes.

Digital Sound

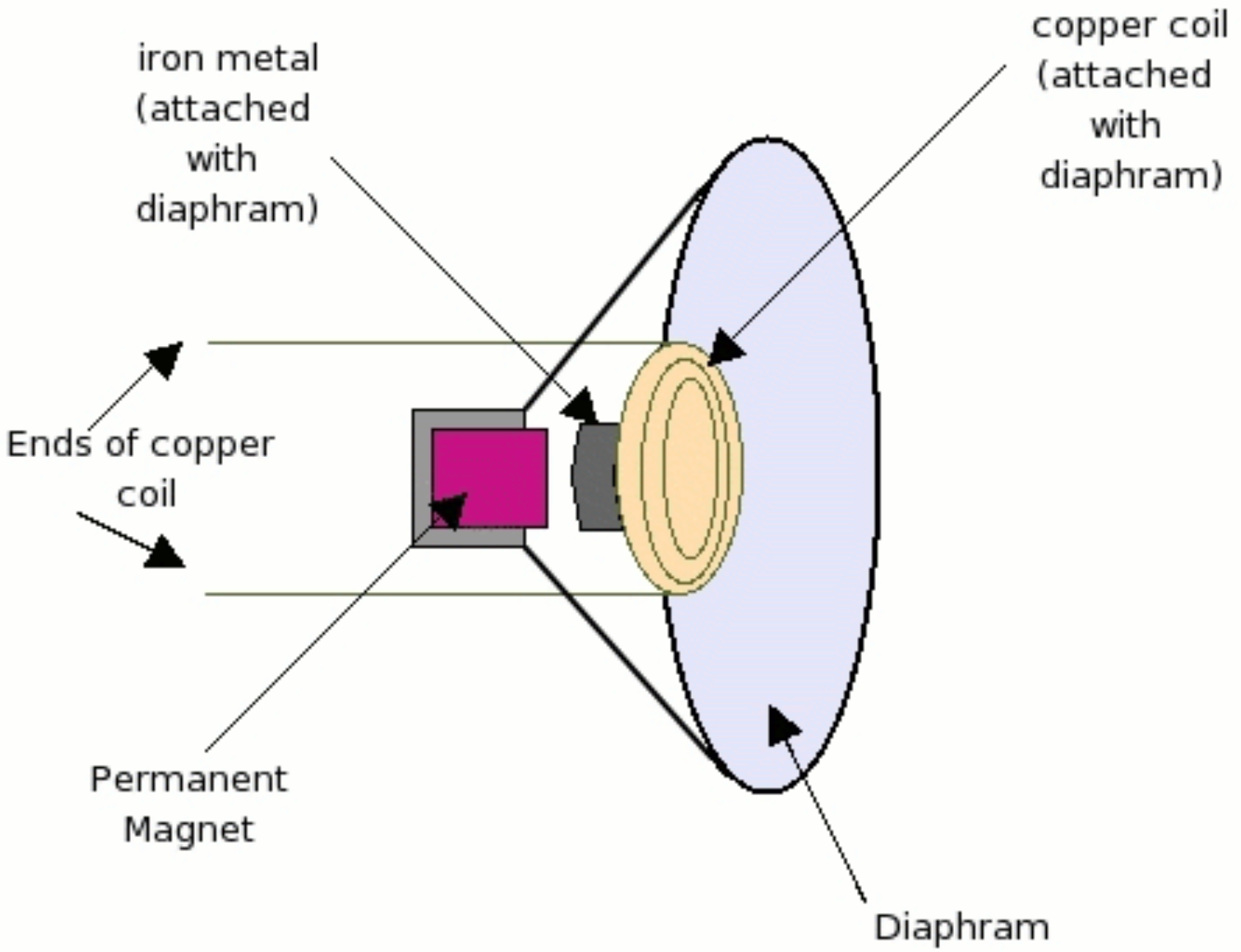
Exercises for reader:

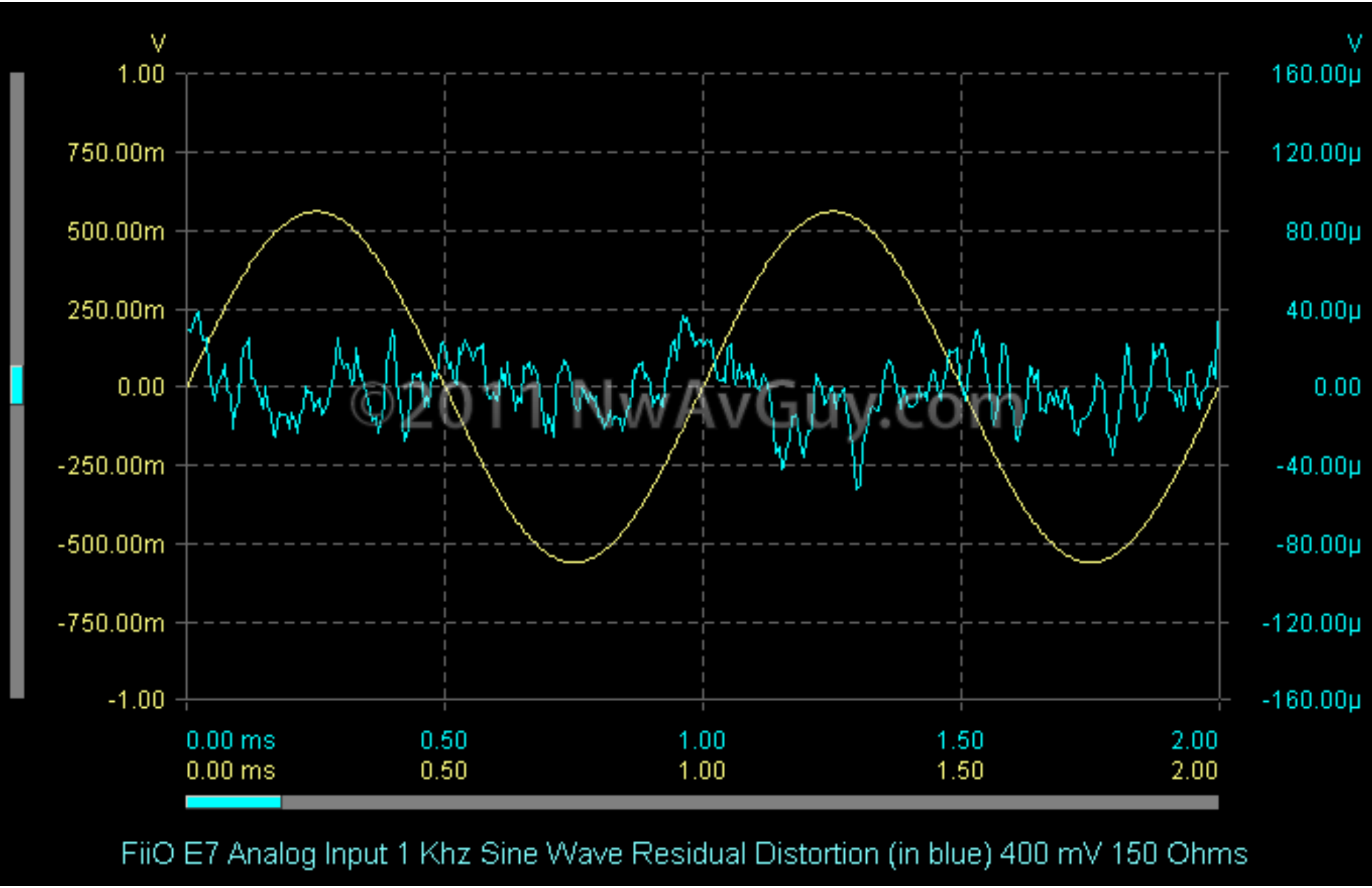
Speakers → Ears → Consciousness

Let's start with speakers

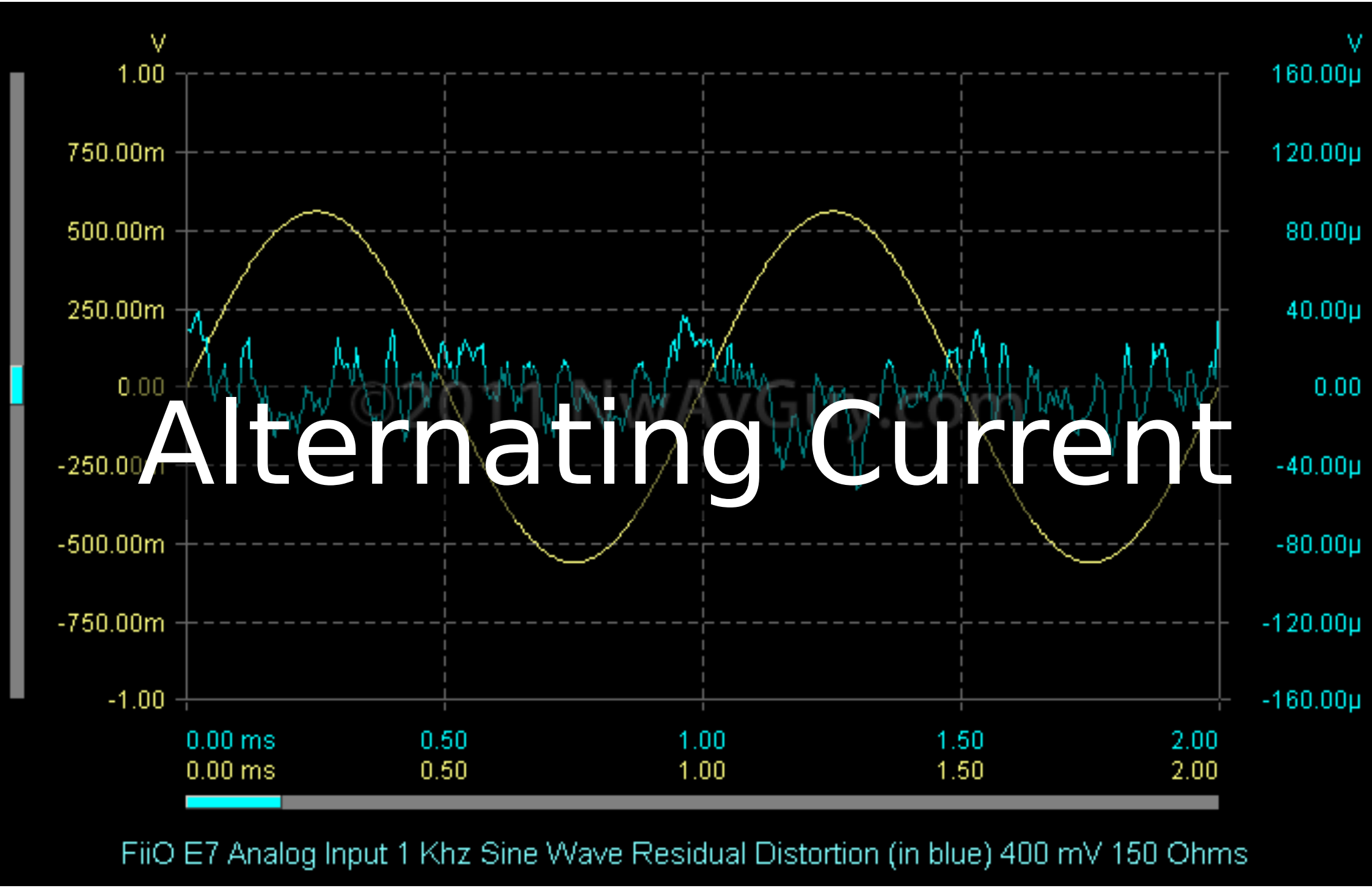

iron metal (attached with diaphram)
copper coil (attached with diaphram)
Ends of copper coil
Permanent Magnet
Diaphram

Sideways is even better


iron metal (attached with diaphram)
copper coil (attached with diaphram)
Ends of copper coil
Permanent Magnet
Diaphram



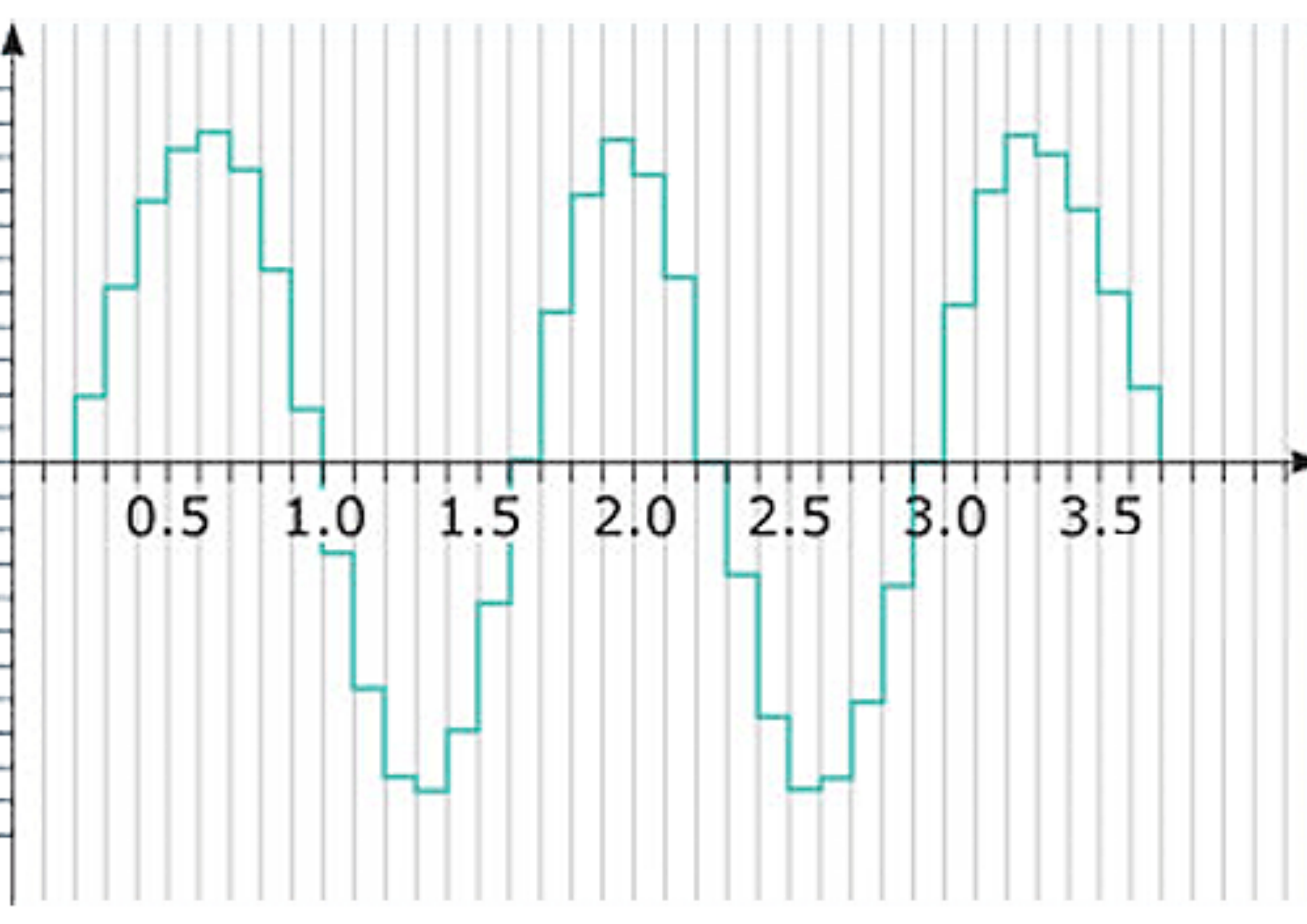
FiiO E7 Analog Input 1 Khz Sine Wave Residual Distortion (in blue) 400 mV 150 Ohms

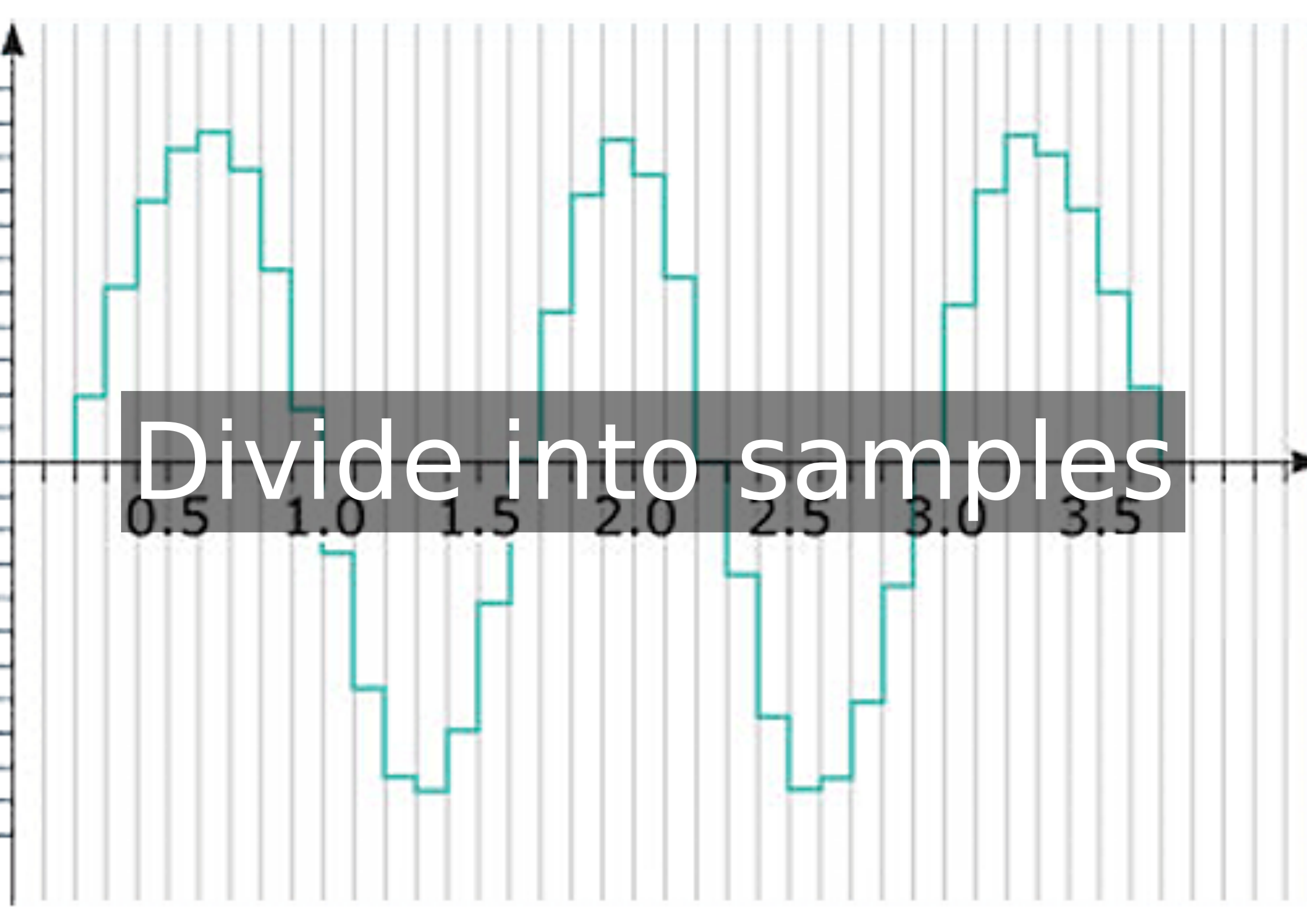

V
1.00
750.00m
500.00m
250.00m
0.00
-250.00m
-500.00m
-750.00m
-1.00
V
160.00µ
120.00µ
80.00µ
40.00µ
0.00
-40.00µ
-80.00µ
-120.00µ
-160.00µ
0.00 ms
0.50
1.00
1.50
2.00
0.00 ms
0.50
1.00
1.50
2.00
Alternating Current
FiiO E7 Analog Input 1 Khz Sine Wave Residual Distortion (in blue) 400 mV 150 Ohms

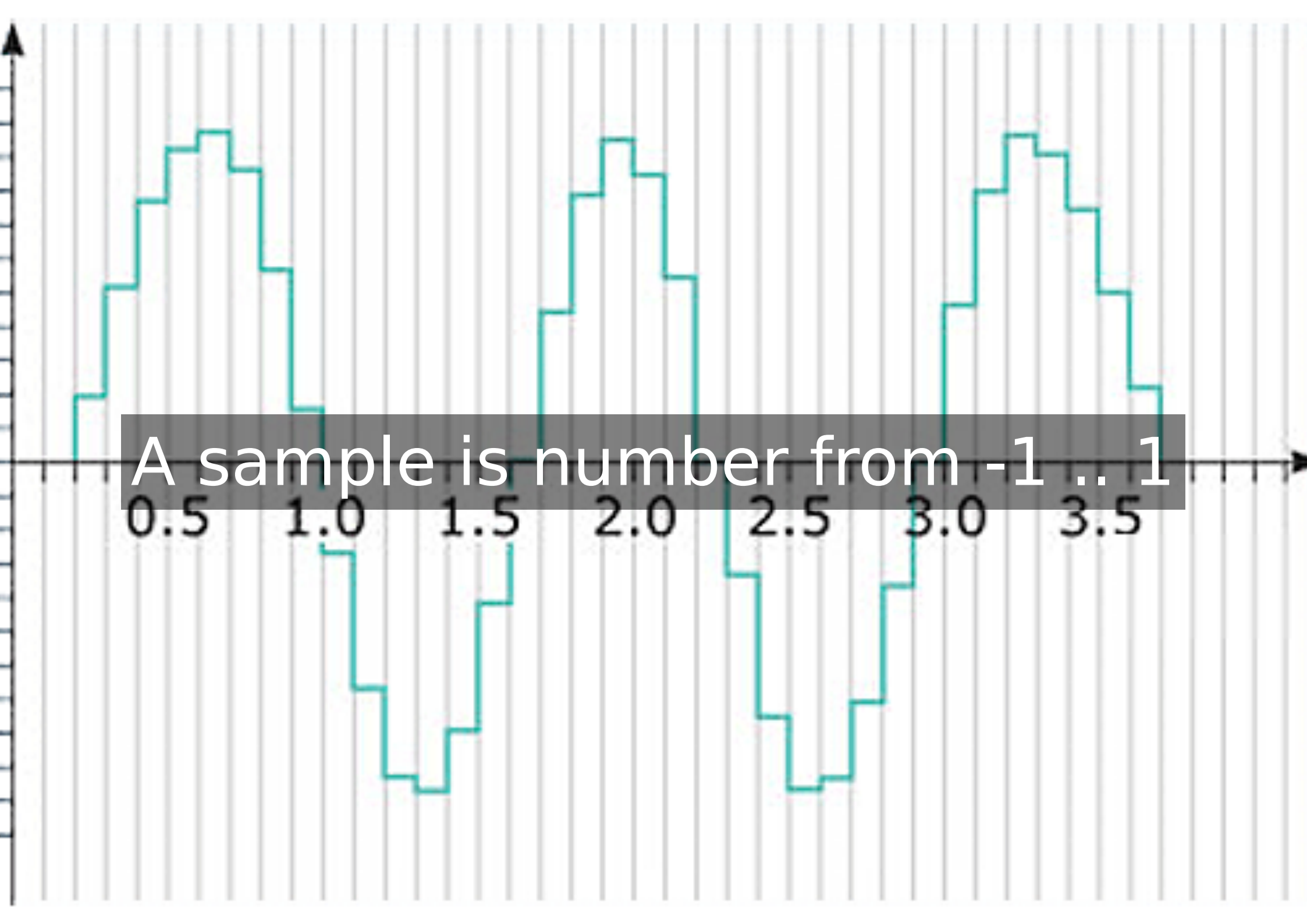
Sound = Speaker movement = Electrical input

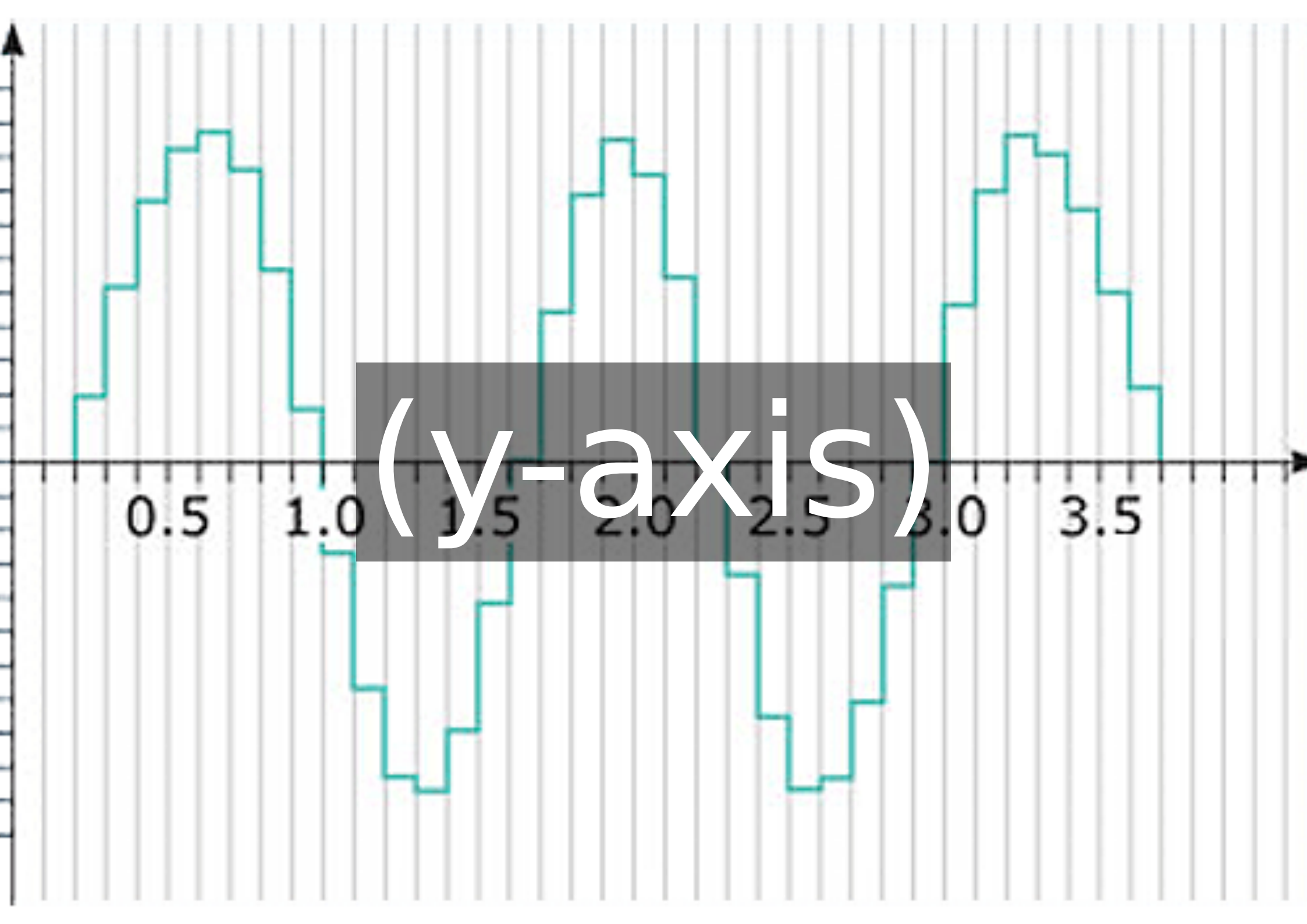

Waves

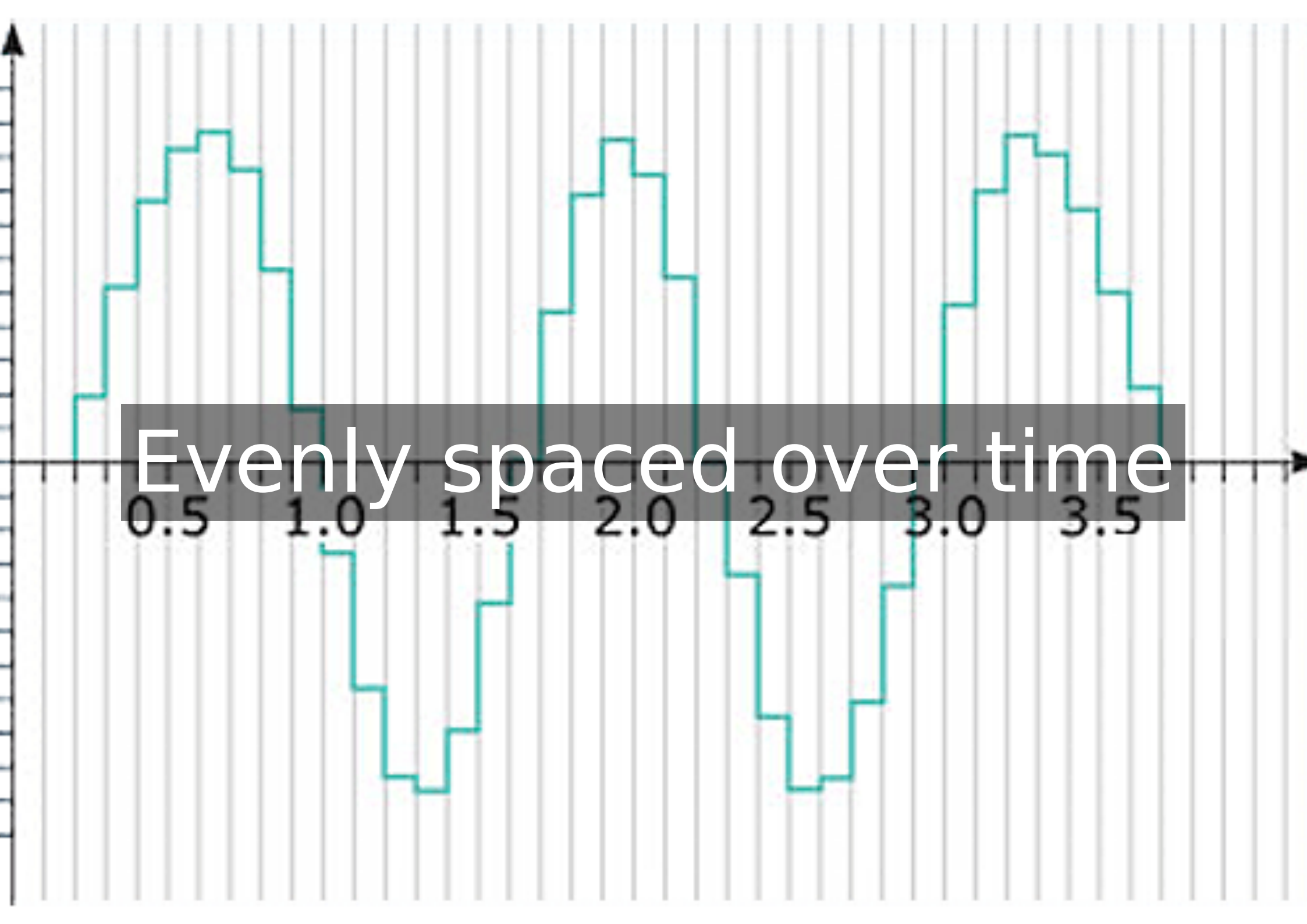
digital?

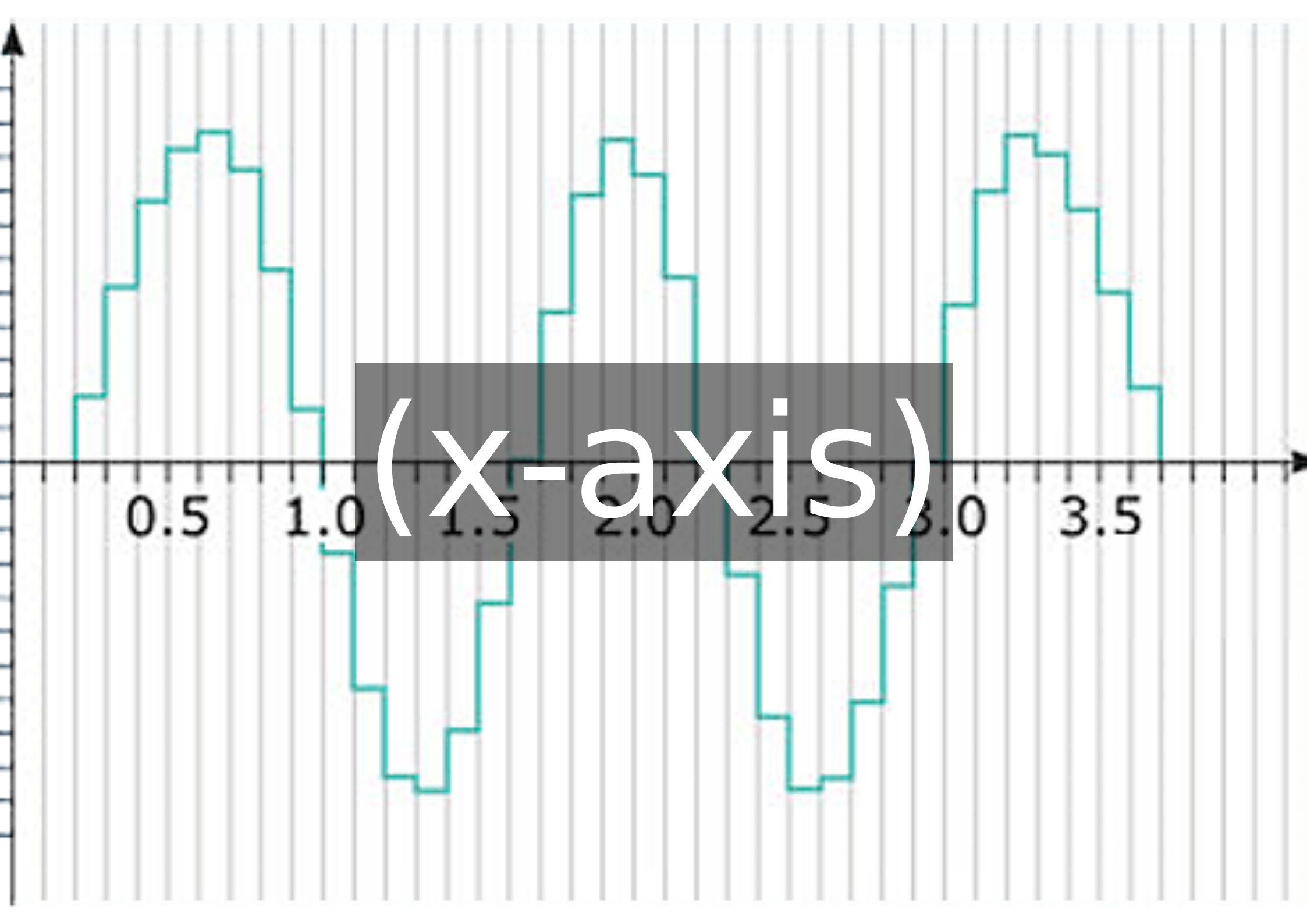

0.5
1.0
1.5
2.0
2.5
3.0
3.5


Divide into samples
0.5
1.0
1.5
2.0
2.5
3.0
3.5


A sample is number from -1 .. 1
0.5
1.0
1.5
2.0
2.5
3.0
3.5


(y-axis)
0.5
1.0
1.5
2.0
2.5
3.0
3.5


Evenly spaced over time
0.5
1.0
1.5
2.0
2.5
3.0
3.5


(x-axis)
0.5
1.0
1.5
2.0
2.5
3.0
3.5

Samples per second

↳ "Sample Rate"

Common sample rates:

DVD: 48,000
CD: 44,100
VOIP: 16,000
Telephone: 8,000

Representation of each sample

↳ "Bit Depth"

8 bit: 0..255 (old video games)

16 bit: 0..65535 (CD)

24 bit: 0..16777215 (DVD-Audio)

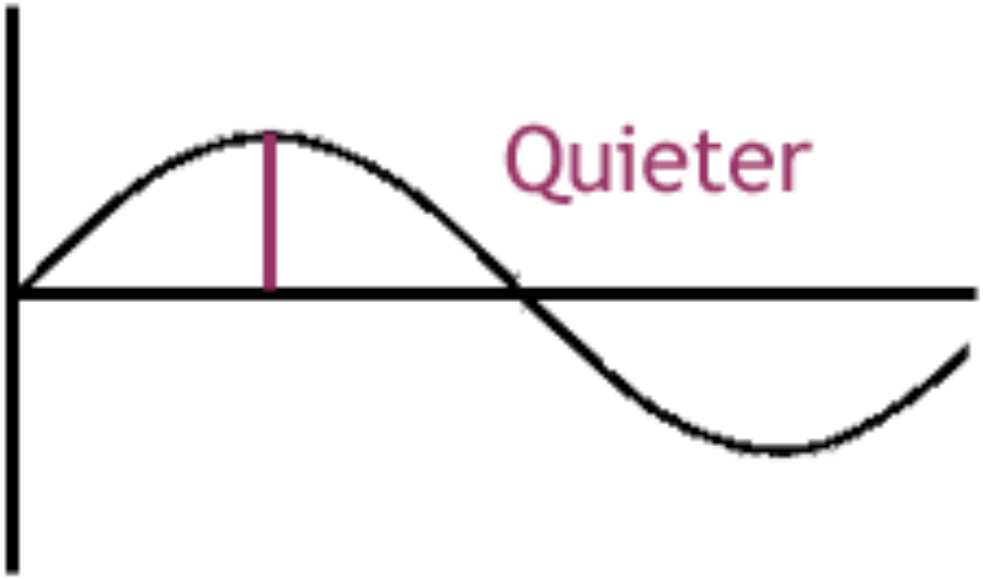
Human Ear: 21 bit @ 40k

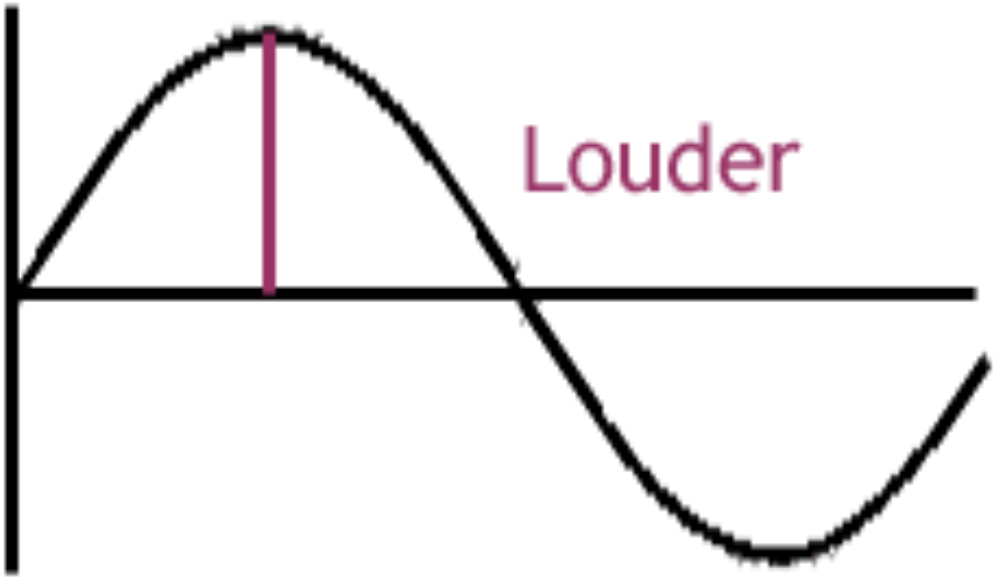
Volume (amplitude) and Frequency

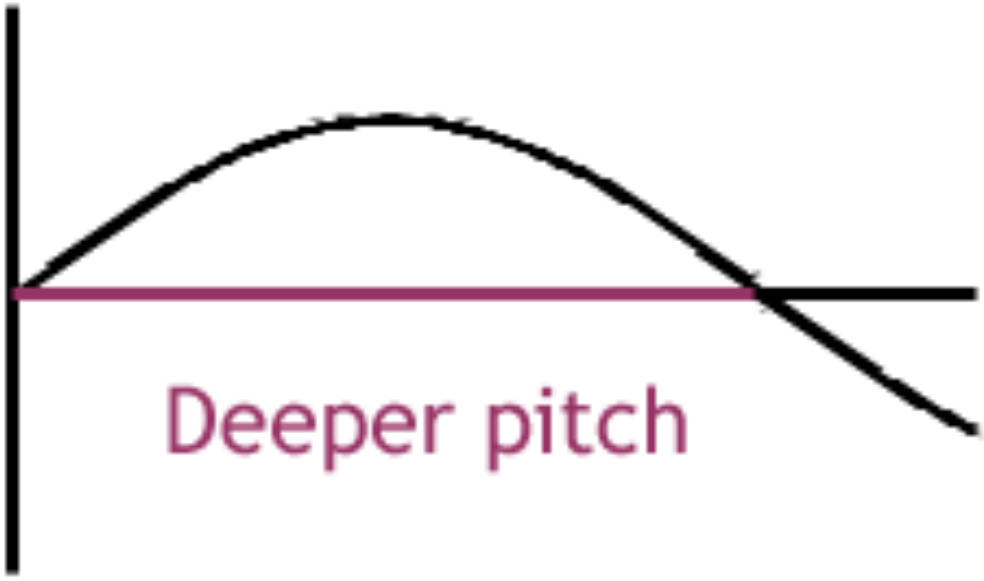
Volume is easy!

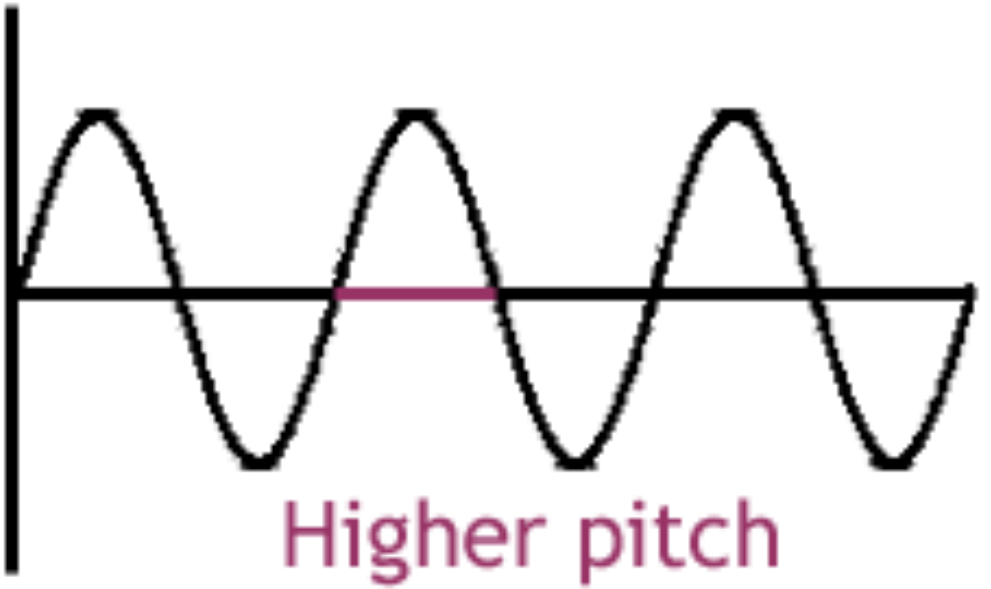
Frequency not so easy

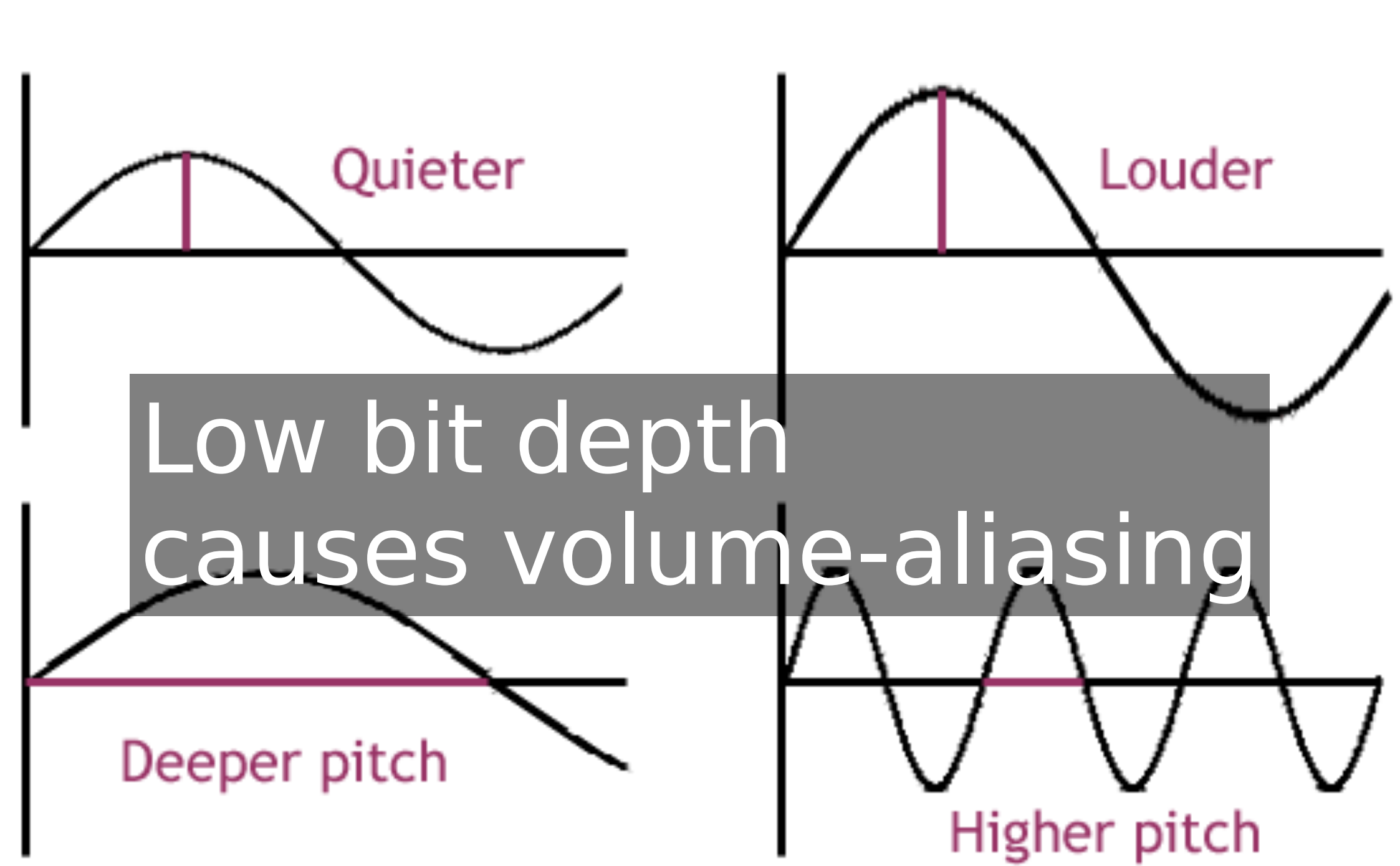
frequency = wiggle speed = pitch

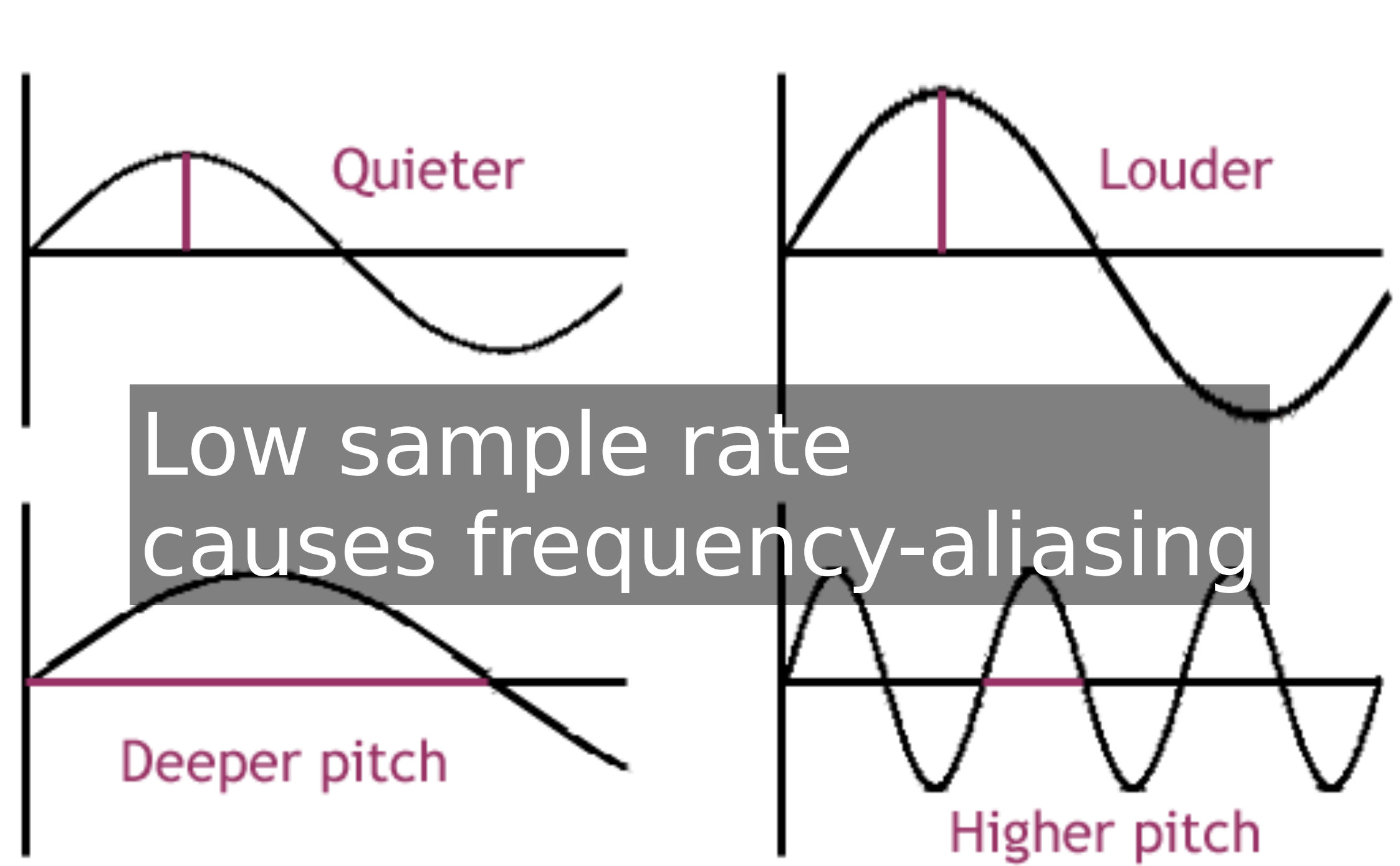


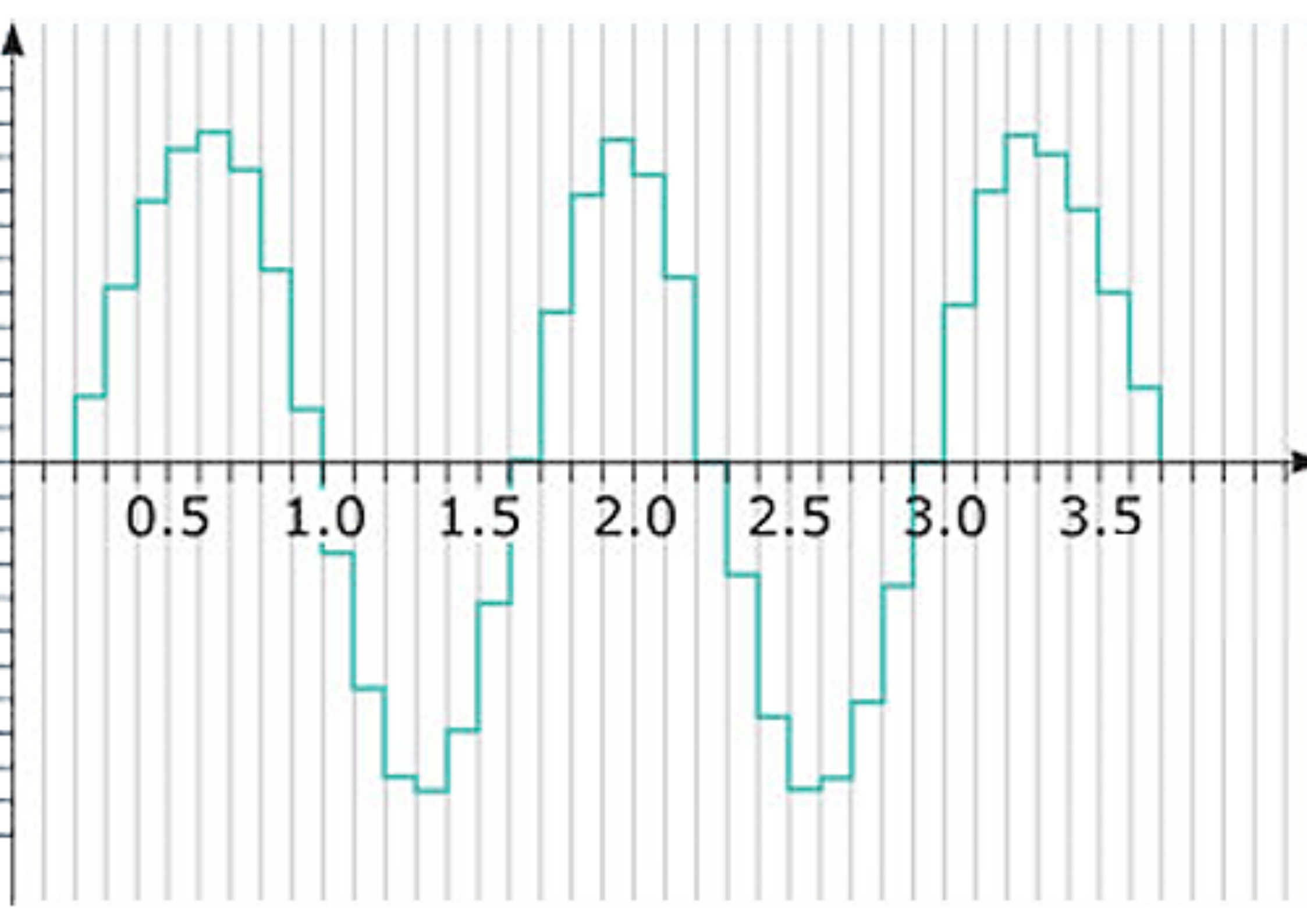







Quieter
Louder
Low bit depth
causes volume-aliasing
Deeper pitch
Higher pitch
© www.science aid.net


Quieter
Louder
Low sample rate
causes frequency-aliasing
Deeper pitch
Higher pitch
© www.science aid.net


0.5
1.0
1.5
2.0
2.5
3.0
3.5

So!

We just gotta generate and play samples!

44100 of them per second!

No problem.

PortAudio

Alternatives: /dev/dsp, aplay, ...

```
PortAudio.init

stream = PortAudio::Stream.open(
  :sample_rate => 44100,
  :frames      => 512,
  :output      => {
    :device        => PortAudio::Device.default_output,
    :channels      => 1,
    :sample_format => :float32
  }
)

stream.start
```

```
buffer = PortAudio::SampleBuffer.new(
  :format   => :float32,
  :channels => 1,
  :frames   => 512
)
```

```
# smallnoise.rb

loop do
  buffer.fill {
    rand()*2 - 1 # From -1 .. 1
  }
  stream << buffer
end
```

Woo... static!

```
sample = rand()*2 - 1
```

```
# smallbeep.rb

time_step = 1 / 48000.0
time = 0.0
loop do
  buffer.fill {
    time += time_step;
    Math.sin( 2 * 3.1415 * time * 440 );
  }
  stream << buffer
end
```

Woo... beeping!

Two beeps at once?

Turns out you just add waves together

```
# smallbeep2.rb

sample  = Math.sin( 2 * 3.1415 * time * 440 );
sample += Math.sin( 2 * 3.1415 * time * 349.23 );
sample /= 2; # Avoid clipping!
```

Let's generalize

```
sample = get_next_sample();
```

Call get_next_sample() over and over

Get a new sample each time

```
def sine {
  Math.sin( 2 * 3.1415 * $time * 440 )
}
```

That was easy.

Not very configurable or dynamic though.

```
# What I want:

sample_gen = sine(440);

# ...

sample = sample_gen.call;
```

This is called a 'generator'

We can make this using a closure!

(aka lambda with bound variables)

```
def sine(freq)
  lambda {
    Math.sin( 2 * 3.1415 * $time * freq );
  }
}
```

```
# So now we have it:

sample_gen = sine(440);

# ... in 'play'

sample = sample_gen.call;
```

```
# Create a 440 Hz sine generator
gen = sine(440);

# Play it!
play( gen );
```

```
play( sine( 440 ) );
```

One more generator tweak

Parameterize generators with generators, and use named params

```
sub sine(freq) {
  freq = genize freq
  lambda {
    Math.sin( 2 * 3.1415 * $time * freq.call );
  }
}
```

```
# Take a parameter and ensure it is a generator.
# If it is already a generator leave it alone,
# otherwise wrap it up so that it *is* a generator.

def genize x
  if x.is_a?(Proc)
    return x
  end
  lambda { x }
end
```

Why would we use generators as parameters?

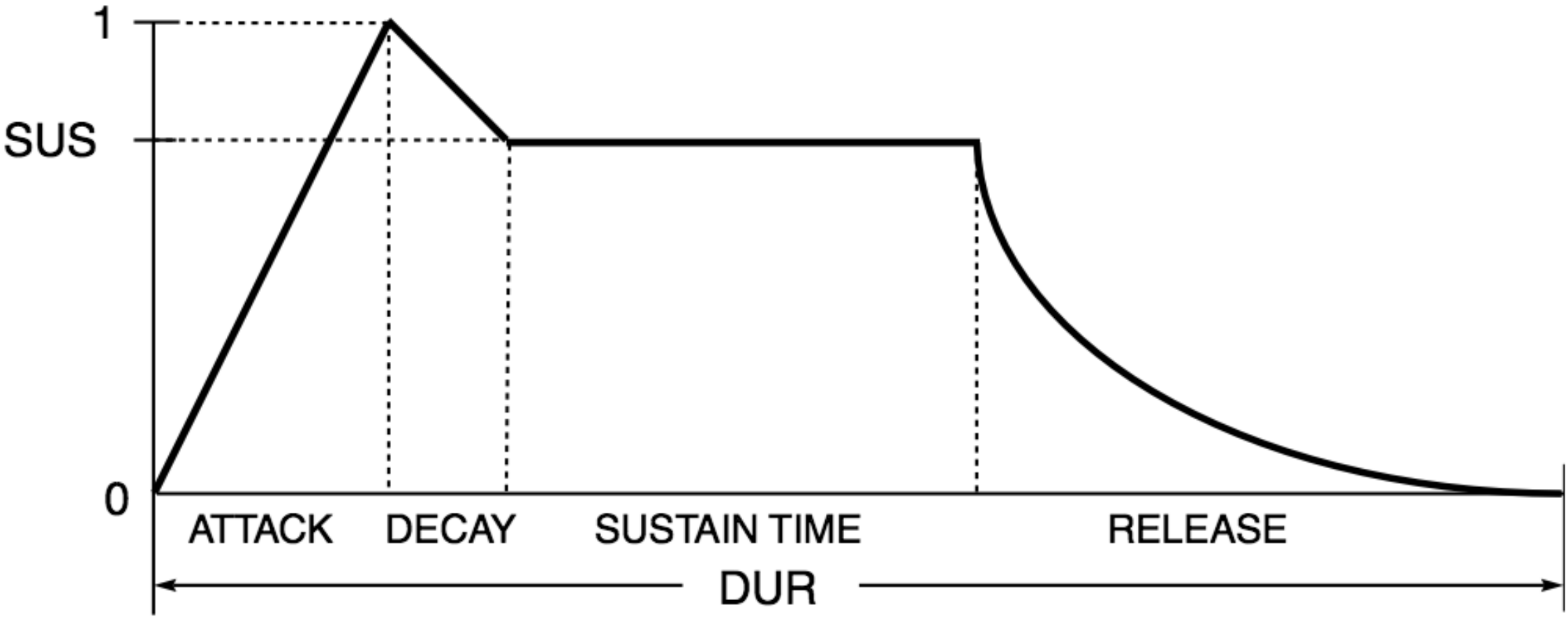
```
lfo = sine(5)
wobble_freq = lambda { lfo.call * 100 + 440 }

play( sine( wobble_freq ) );
```

Now we're cooking with FIRE!

Plays forever...

Envelope Generator

Attack, [Decay], Sustain, Release


1
SUS
0
ATTACK
DECAY
SUSTAIN TIME
RELEASE
DUR

```
envelope( gen, attack, sustain, release )

# For example

envelope( sine(440), 2, 0, 2 )
```

returns nil when done

Sequence Generator

```
seq( gens )

# For example

play(
  seq([
    envelope(square(440), 2, 0, 2),
    envelope(square(220), 2, 0, 2),
  ])
)
```

Simultaneous Generators Generator

```
play(
  sum([
    envelope(square(440), 2, 0, 2),
    envelope(square(220), 2, 0, 2),
    envelope(square(880), 2, 0, 2),
    envelope(square(660), 2, 0, 2),
  ])
)
```

Let's build something we can PLAY

Input Control Generator

Using my touch-screen

```
$ xmousepos
838 574 221 170
```

```
x = `xmousepos`.split[0];
```

```
def mousefreq()
  count = 0
  x = 0.0
  lambda {
    count += 1
    if count  % 1000 == 0
      x = `xmousepos`.split[0].to_f
    end
    x
  }
end
```

```
play(
   amp(
      sine( mousefreq() ),
      mousevol()
   )
);
```

And now we have a synth :)

THE END

References

Source Code:
http://thelackthereof.org/NoiseGen
http://github.com/awwaiid/ruby-noise

Digital fidelity discussion:
http://people.xiph.org/~xiphmont/demo/neil-young.html

BONUS SLIDES

Note / Song Generators

```
note( 'A4' )
```

```
segment( 'A4 F3' ),
```

```
combine([
    segment( 'A4' ),
    segment( 'F3' ),
])
```

Formula-Based Noise
http://countercomplex.blogspot.com/2011/10/algorithmic-symphonies-from-one-line-of.html