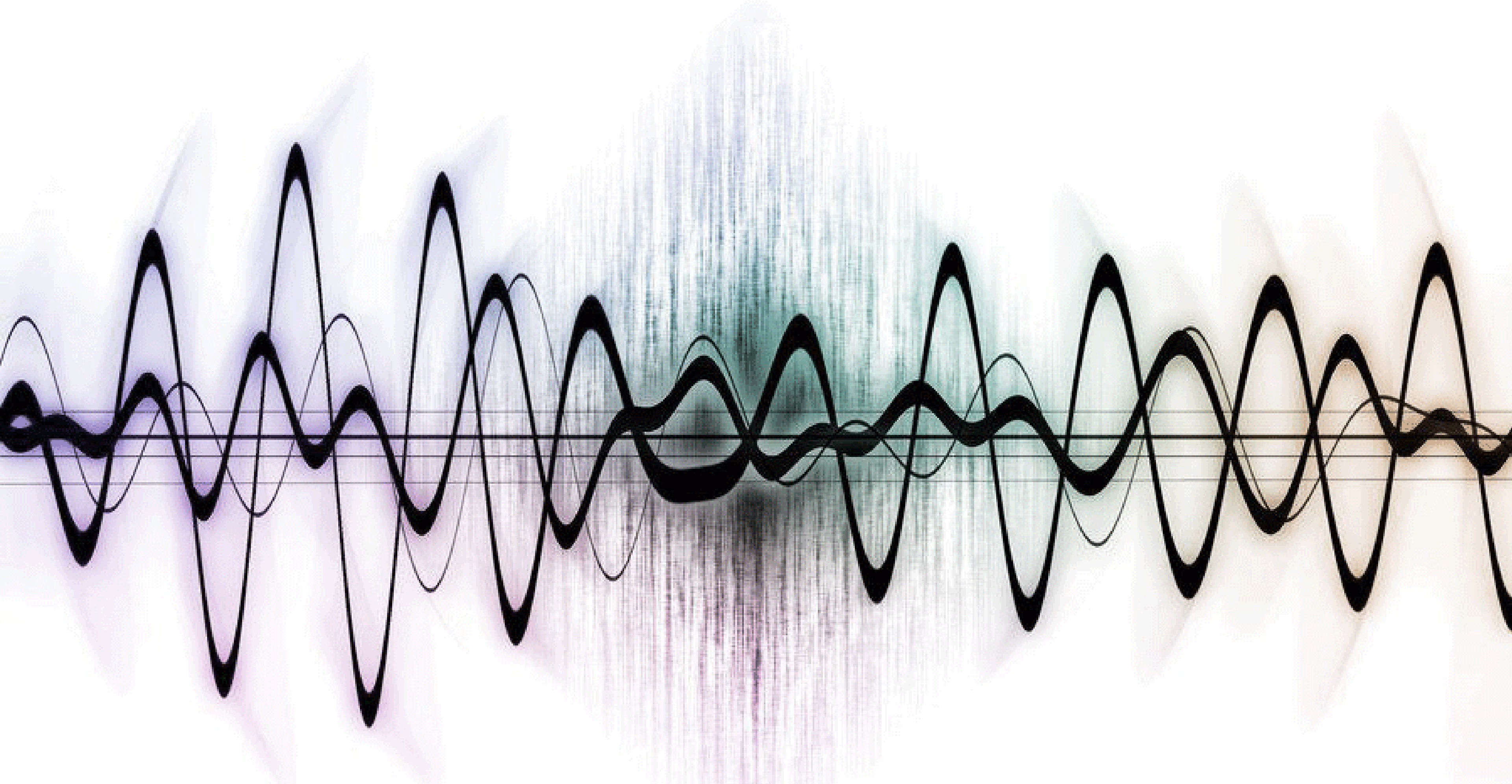
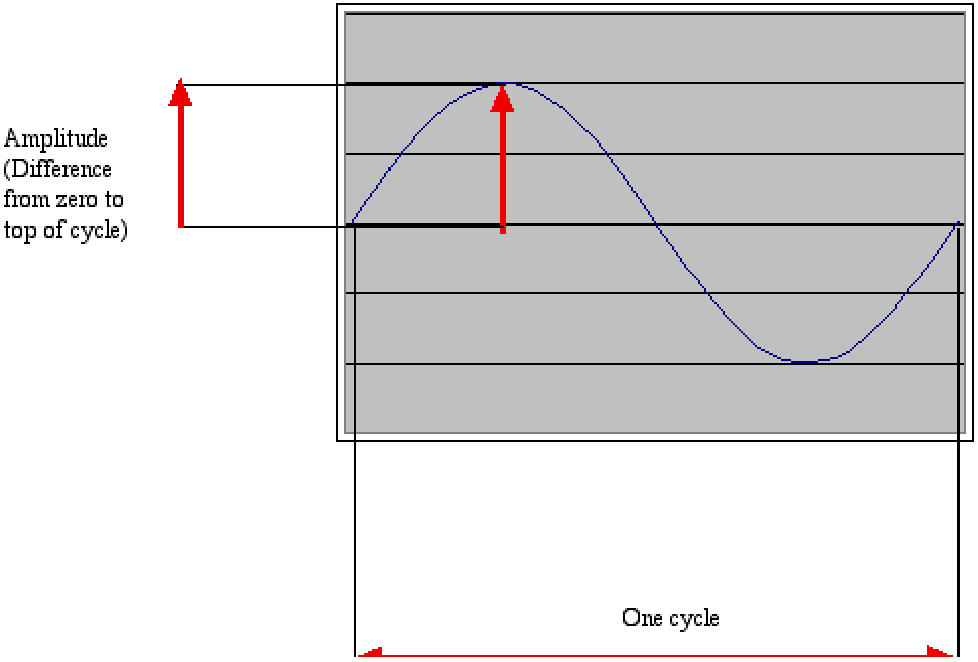


Tinkering Audio II: Further Notes on Digital Sound

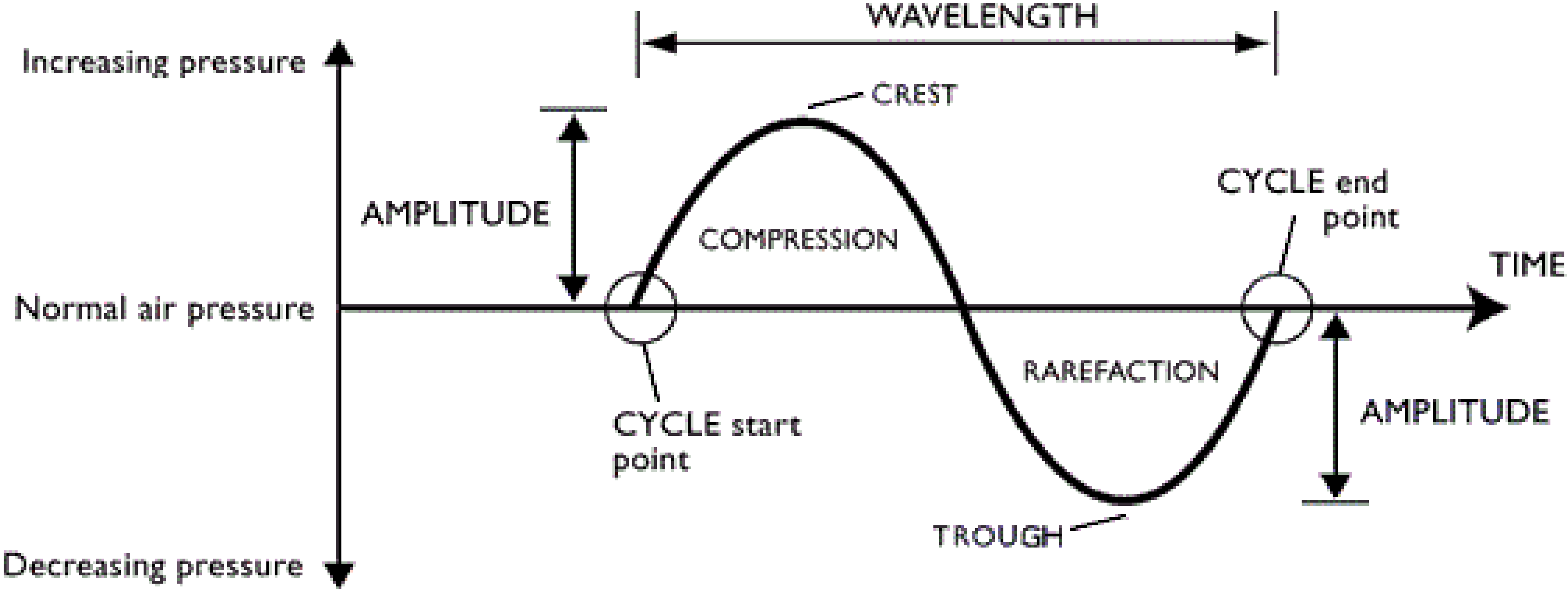
Creative Computing: Tinkering – Lecture 9 – Michael Scott

Recap on Last Week

- Sounds are waves of air pressure
  - Sound comes in cycles
  - The *frequency* of a wave is the number of cycles per second (cps), or *Hertz*
  - Complex sounds have more than one frequency in them.



Recap on Last Week



Recap on Last Week

```python
OUTPUT_FILENAME = "noise.wav"
LENGTH_OF_FILE_IN_SECONDS = 1
CHANNEL_COUNT = 1
SAMPLE_WIDTH = 2                                                    # 2 bytes per sample
SAMPLE_RATE = 44100
SAMPLE_LENGTH = SAMPLE_RATE * LENGTH_OF_FILE_IN_SECONDS
COMPRESSION_TYPE = 'NONE'
COMPRESSION_NAME = 'not compressed'
MAX_VALUE = 32767
FREQUENCY = 15000

import wave
import struct
import math

noise_out = wave.open(OUTPUT_FILENAME, 'w')
noise_out.setparams((CHANNEL_COUNT, SAMPLE_WIDTH, SAMPLE_RATE,
    SAMPLE_LENGTH, 'NONE', 'not compressed'))

values = []

for i in range(0, SAMPLE_LENGTH):
    value = math.sin(2.0 * math.pi * FREQUENCY * (i / SAMPLE_RATE)) \
        * MAX_VALUE
    packed_value = struct.pack('h', int(value))

    for j in range(0, CHANNEL_COUNT):
        values.append(packed_value)


noise_out.writeframes(b''.join(values))
noise_out.close()
```

Recap on Last Week

```
OUTPUT_FILENAME = "noise.wav"
LENGTH_OF_FILE_IN_SECONDS = 1
CHANNEL_COUNT = 1
SAMPLE_WIDTH = 2                # 2 bytes per sample
SAMPLE_RATE = 44100
SAMPLE_LENGTH = SAMPLE_RATE * LENGTH_OF_FILE_IN_SECONDS
COMPRESSION_TYPE = 'NONE'
COMPRESSION_NAME = 'not compressed'
MAX_VALUE = 32767
FREQUENCY = 15000
```

Recap on Last Week

```
noise_out = wave.open(OUTPUT_FILENAME, 'w')
noise_out.setparams((CHANNEL_COUNT, SAMPLE_WIDTH, SAMPLE_RATE,
    SAMPLE_LENGTH, 'NONE', 'not compressed'))

values = []

for i in range(0, SAMPLE_LENGTH):
    value = math.sin(2.0 * math.pi * FREQUENCY * (i / SAMPLE_RATE)) \
        * MAX_VALUE
    packed_value = struct.pack('h', int(value))

    for j in range(0, CHANNEL_COUNT):
        values.append(packed_value)


noise_out.writeframes(b''.join(values))
noise_out.close()
```

Tinkering Audio

TONE PERCEPTION

Tone Perception

- It's strange, but many aspects our perception relies on ratios of difference rather than absolute values.
  – We hear changes between 200 Hz and 400 Hz, as the same as changes between 500 Hz and 1000 Hz (1:2)
  – Similarly, 200 Hz to 600 Hz, and 1000 Hz to 3000 Hz, etc. (1:3)
  – Repeatedly multiplying amplitude by a constant factor is perceived by humans as a series of equal increases.
- Intensity (volume) is measured as watts per meter squared
  – A change from 0.1W/m2 to 0.01 W/m2, "sounds" the same to us as 0.001W/m2 to 0.0001W/m2

Tone Perception

- We experience sound in a "logarithmic" way
- Our perception of volume is related (logarithmically) to changes in amplitude
  - If the amplitude doubles, it's about a 3 decibel (dB) change
- Our perception of pitch is related (logarithmically) to changes in frequency
  - Higher frequencies are perceived as higher pitches
  - We can hear between 5 Hz and 20,000 Hz (20 kHz)
  - Middle C is 440 Hz

“Logarithmically?”

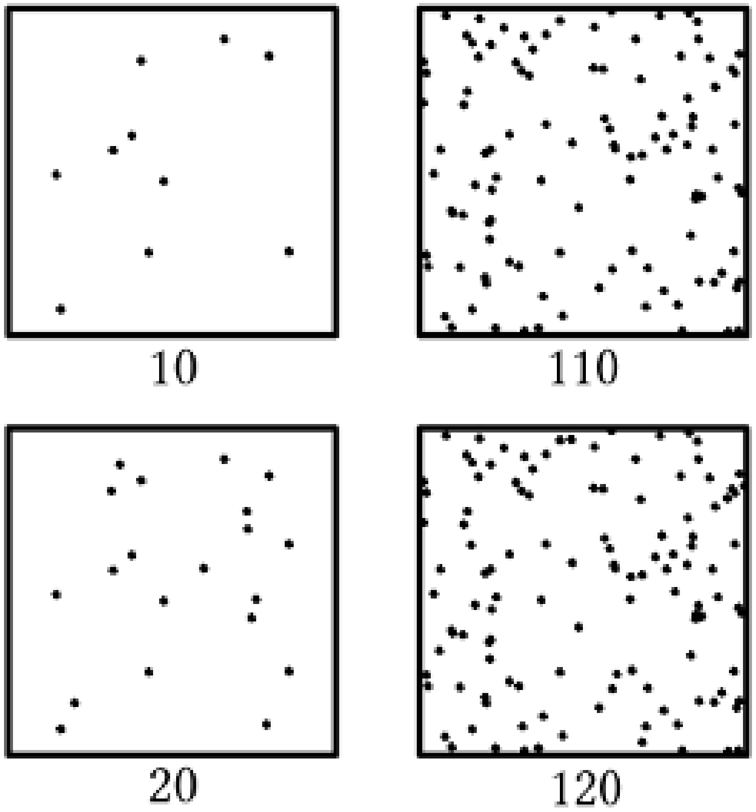
- A logarithm represents the power to which a fixed number (the base) must be raised to produce a given number
- It answers the question:
  - How many of this number do we multiply to get that number
- For example:
  - $\mathrm{Log}_2(8) = 3$
  - Because: $2 * 2 * 2 = 8$
- It represents the opposite of powers: $2^3 = 8$

“Logarithmically?”

- This can be expressed a ratio between two intensities, for example the Decibel:
  $10 * \log_{10}(I_1/I_2)$
  – As an absolute measure, it's in comparison to threshold of audibility
  – 0 dB can't be heard.
  – Normal speech is 60 dB.
  – A shout is about 80 dB
  – A music concert can be around 100-120 dB

“Logarithmically?”

The Weber-Fetcher Law:



“Logarithmically?”

- Although in absolute terms, the difference between the upper cells and the lower cells are the same, humans have trouble perceiving the difference
- This is because perception of change is non-linear
- Tends to happen with our experience of pitch, with absolutely identical tones (in reality) being “perceived” differently
  – For example, Shepard tones

Shepard Tones

http://www.moillusions.com/wp-content/uploads/2006/05/shepards.mp3

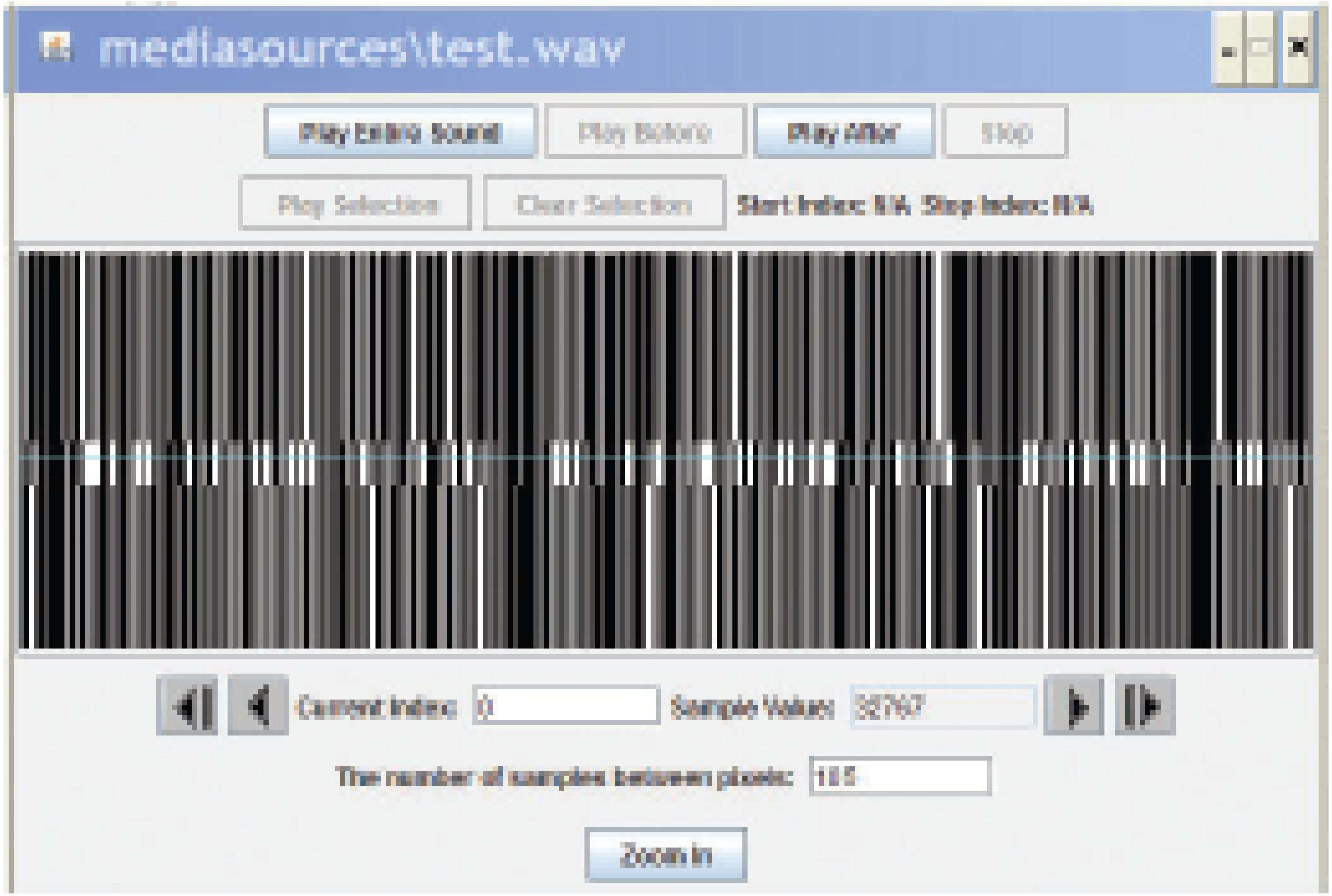
Tinkering Audio

TONAL LIMITATION

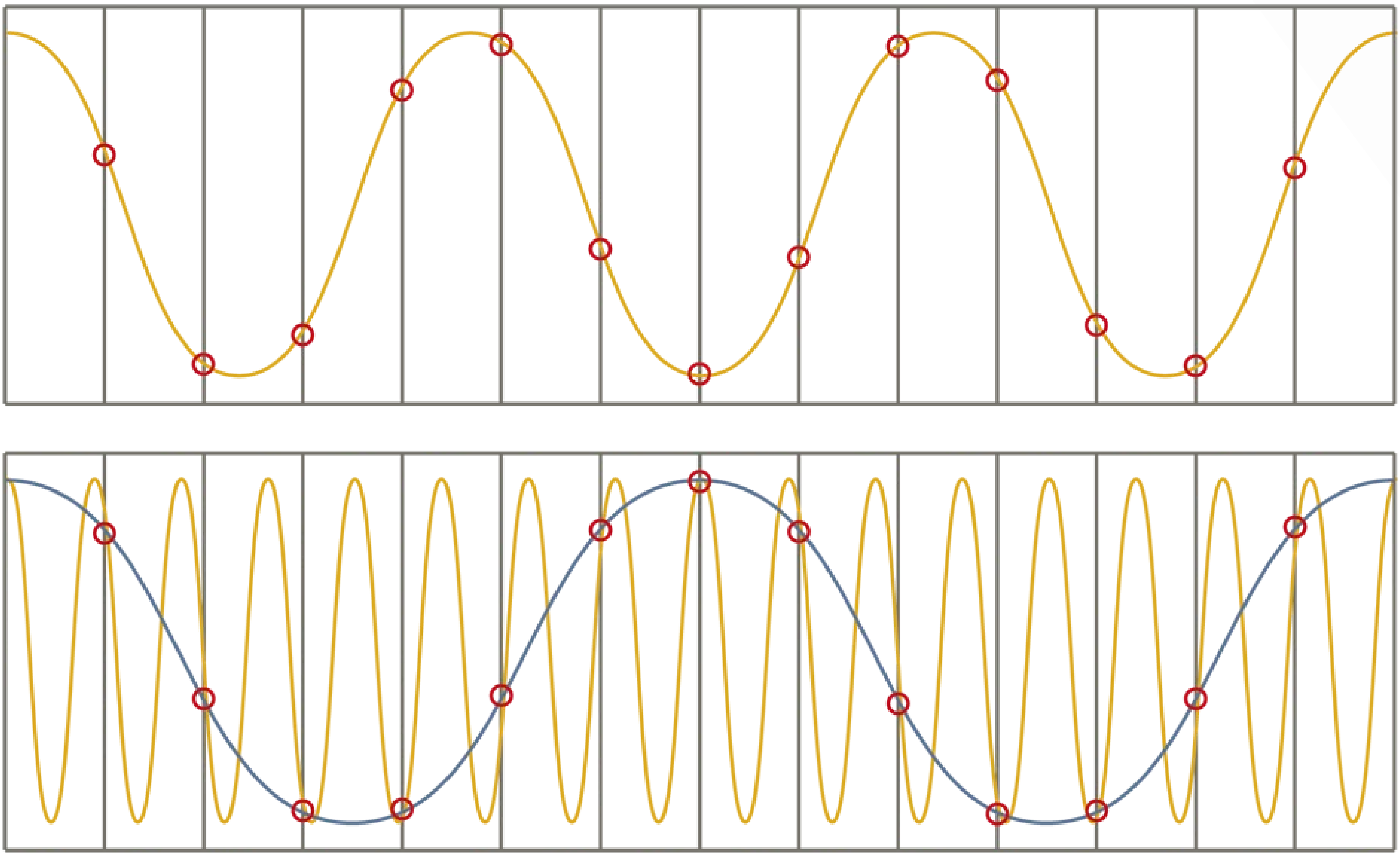
Tonal Limitations

- Analogue waves cannot be fully reproduced by digital systems and hardware
- Computers are limited by the range of numbers that they represent
  - Thus, the distinction between different amplitudes is limited and leads to a problem known as “clipping”
- Computers are also limited by the number of elements that represent a unit of time
  - Thus, the ability to capture signals is limited and leads to a problem known as “aliasing”

Clipping


mediasources\test.wav
Play Entire Sound
Play Before
Play After
Stop
Play Selection
Clear Selection
Start Index: N/A Stop Index: N/A
Current Index:
0
Sample Value:
32767
The number of samples between pixels:
Zoom In

Aliasing



Tonal Limitations

- We address "clipping" through normalization
  - "the application of a constant amount of gain to an audio recording to bring the average or peak amplitude to a target level (the norm)"
- We address "aliasing" through the application of Nyquist theorem
  - Thus, the ability to capture signals is limited and leads to a problem known as "aliasing"

Clipping: Normalizing Sound

```
def normalize(sound):
    largest = 0
    for s in getSamples(sound):
        largest = max(largest, getSampleValue(s))
    amplification = 32767.0 / largest

    print "Largest sample value in original sound was",  largest
    print ”Amplification multiplier is", amplification

    for s in getSamples(sound):
        louder =  amplification * getSampleValue(s)
        setSampleValue(s, louder)
```

This loop finds the loudest sample

Why 32,767.0 ?

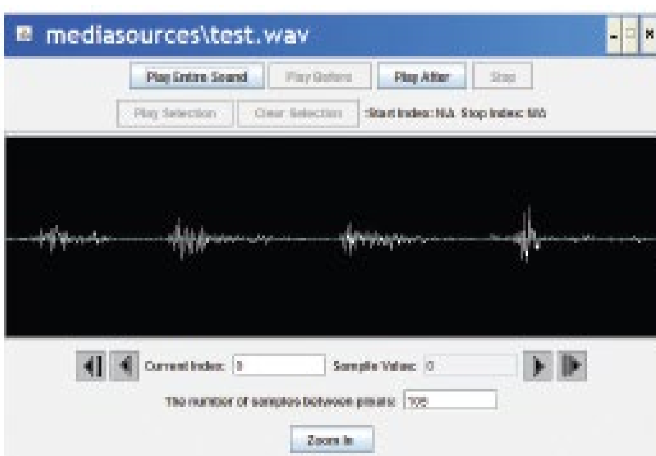
This loop actually amplifies the sound

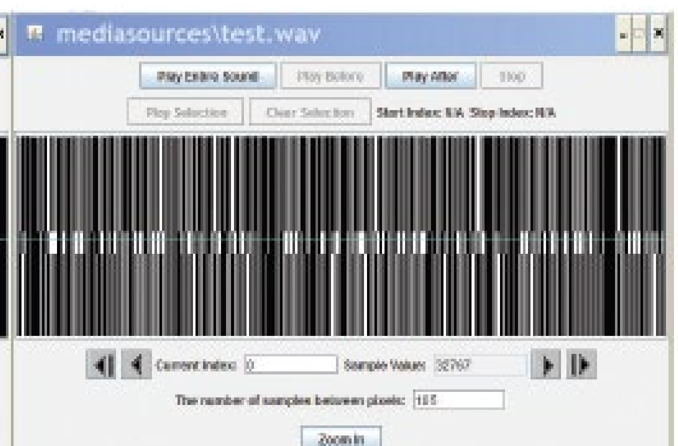
Clipping: Extreme Values

```
def maximize(sound):
 for sample in getSamples(sound):
  value = getSampleValue(sample)
  if value > 0:
   setSampleValue(sample, 32767)
  if value < 0:
   setSampleValue(sample, -32768)
```

What happens when we maximise a sound?

- All samples over 0: Make it 32767
- All samples at or below 0: Make it -32768
- Can we still hear speech?





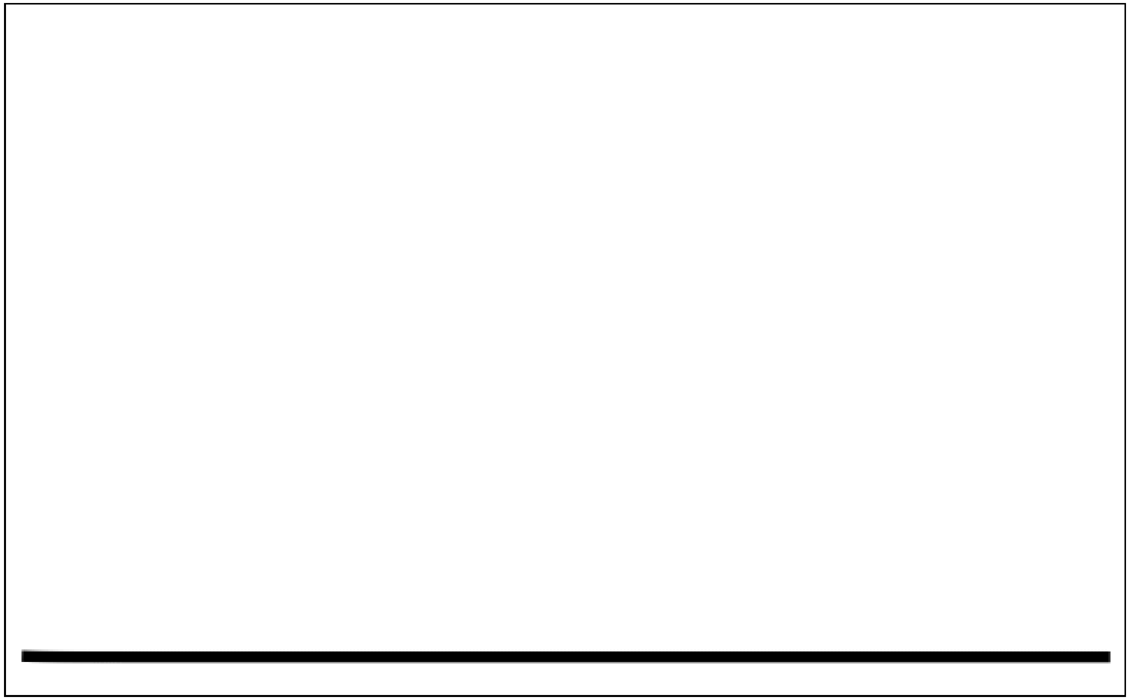
Aliasing: Nyquist Theorem

- We need twice as many samples as the maximum frequency in order to represent (and recreate, later) the original sound.
- The number of samples recorded per second is the sampling rate
  - If we capture 8000 samples per second, the highest frequency we can capture is 4000 Hz
    - That's how phones work
  - If we capture more than 44,000 samples per second, we capture everything that we can hear (max 22,000 Hz)
    - CD quality is 44,100 samples per second

Tinkering Audio

TONE COMBINATION

Superposition

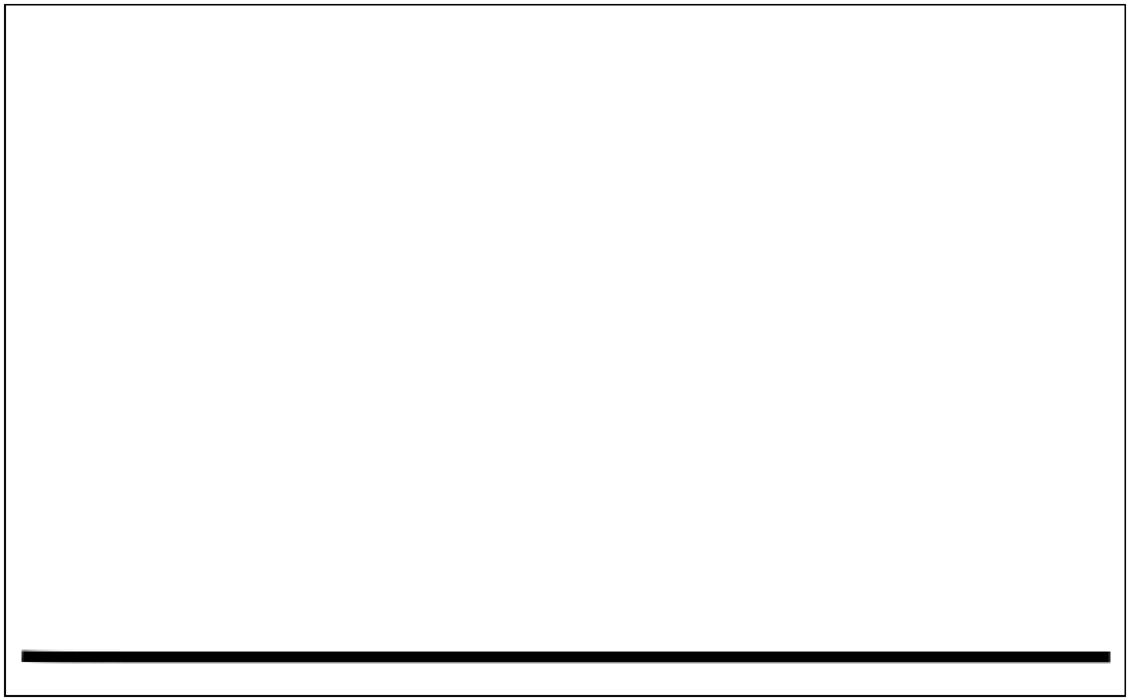


- Superposition applies to waves whenever two (or more) are travelling through the same medium at the same time.

Superposition

- The waves pass through each other without being disturbed. However, the displacement of the medium changes.

Superposition



- The net displacement of the medium at any point in space or time, is simply the sum of the individual wave displacements.

Superposition

- (Note: In reality, not all waves behave like this)

Fourier and Superposition

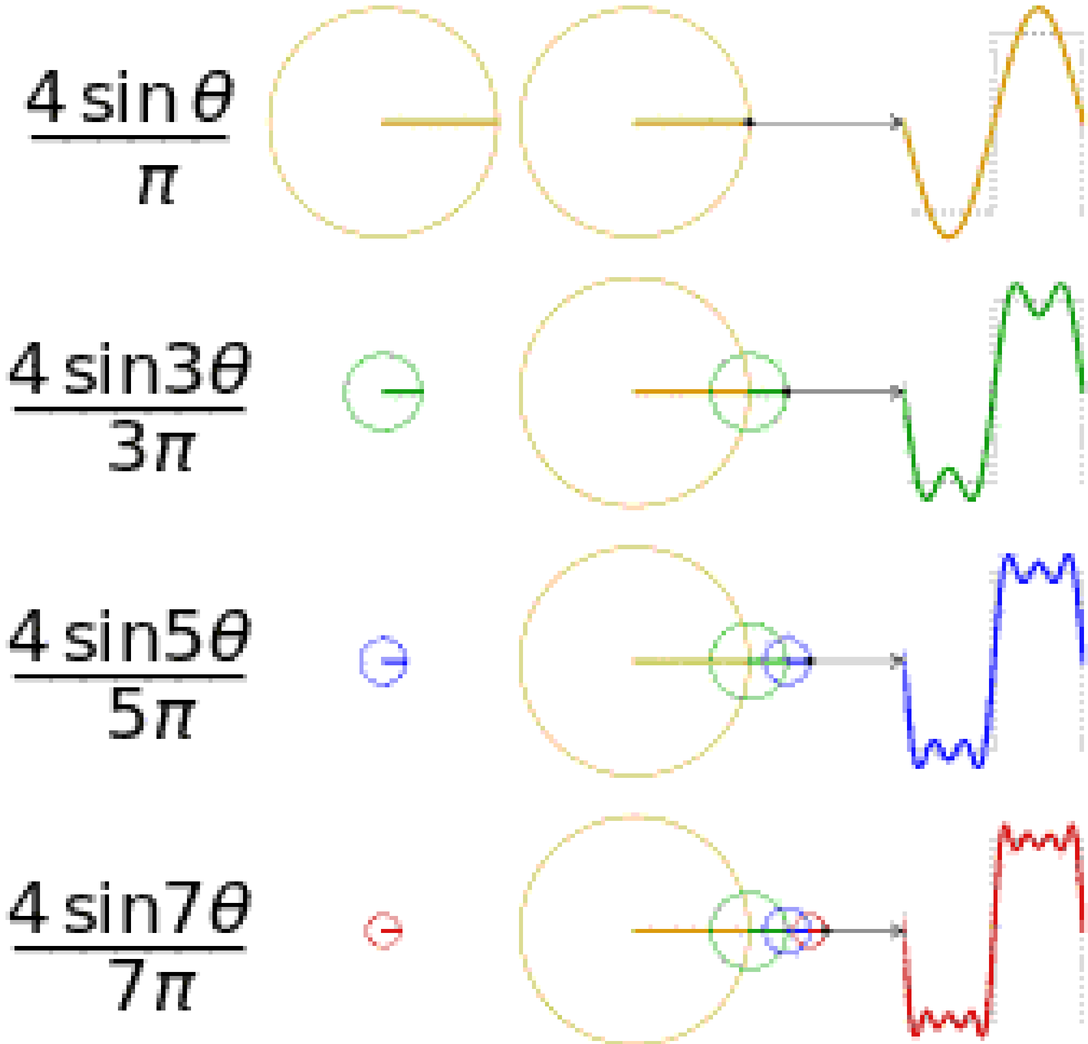
- Prior to Fourier's work, no solution to a particular complex equation was known in the general case, although particular solutions were known if the variables behaved in a simple way; in particular, following the sine or cosine functions.

- These simple solutions are now sometimes called “eigensolutions”.

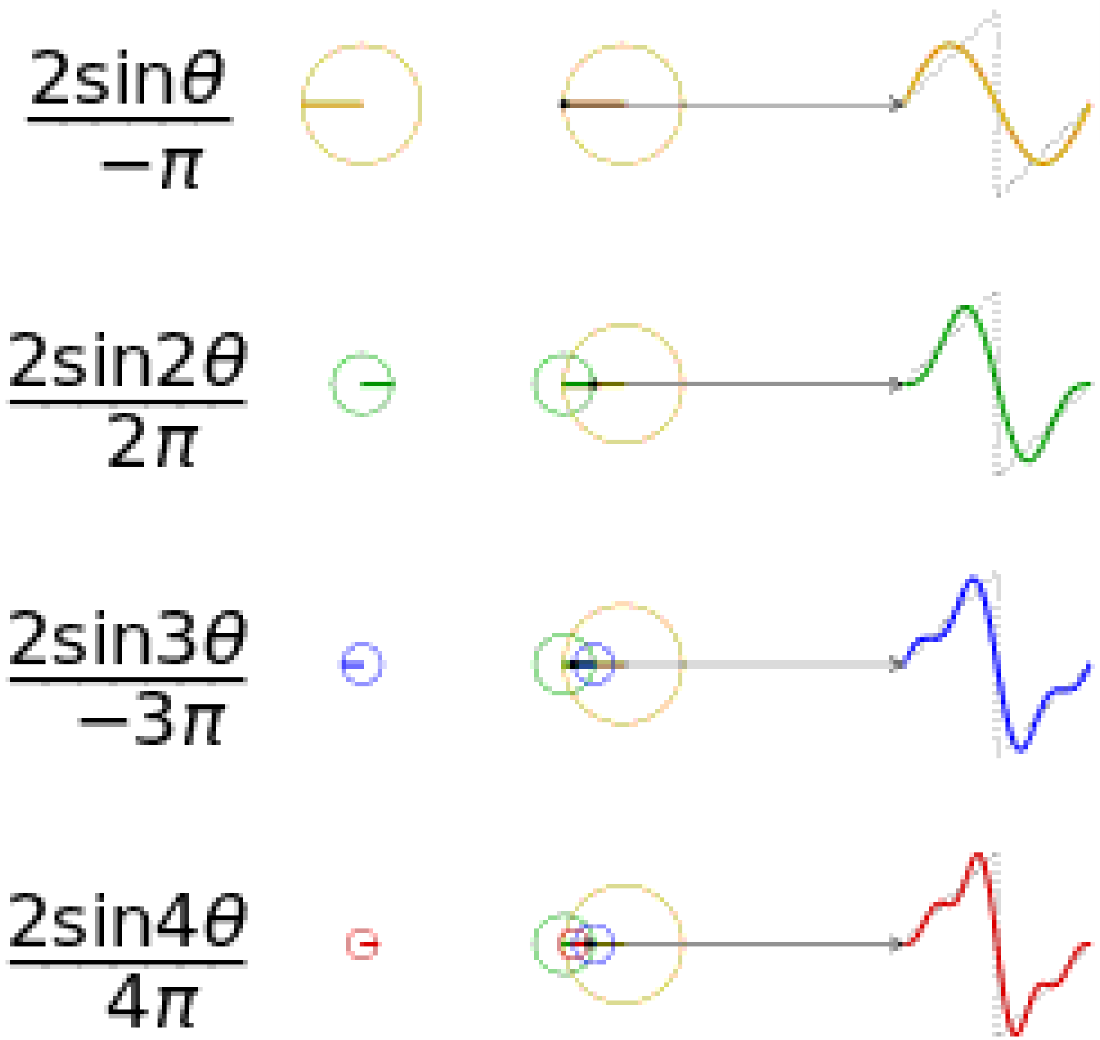
Fourier and Superposition

- Fourier's idea was to model a complicated variable as a superposition (or linear combination) of simple sine and cosine waves, and then to write the solution as a superposition of the corresponding eigensolutions.

- This superposition (or linear combination) is now called the Fourier series. Correspondingly, Fourier series' can be used in the construction of complex tones.



Approximating a Square Wave



Approximating a Saw-Tooth Wave

Other Waves

There are other ways to calculate these waves using the appropriate formulae:

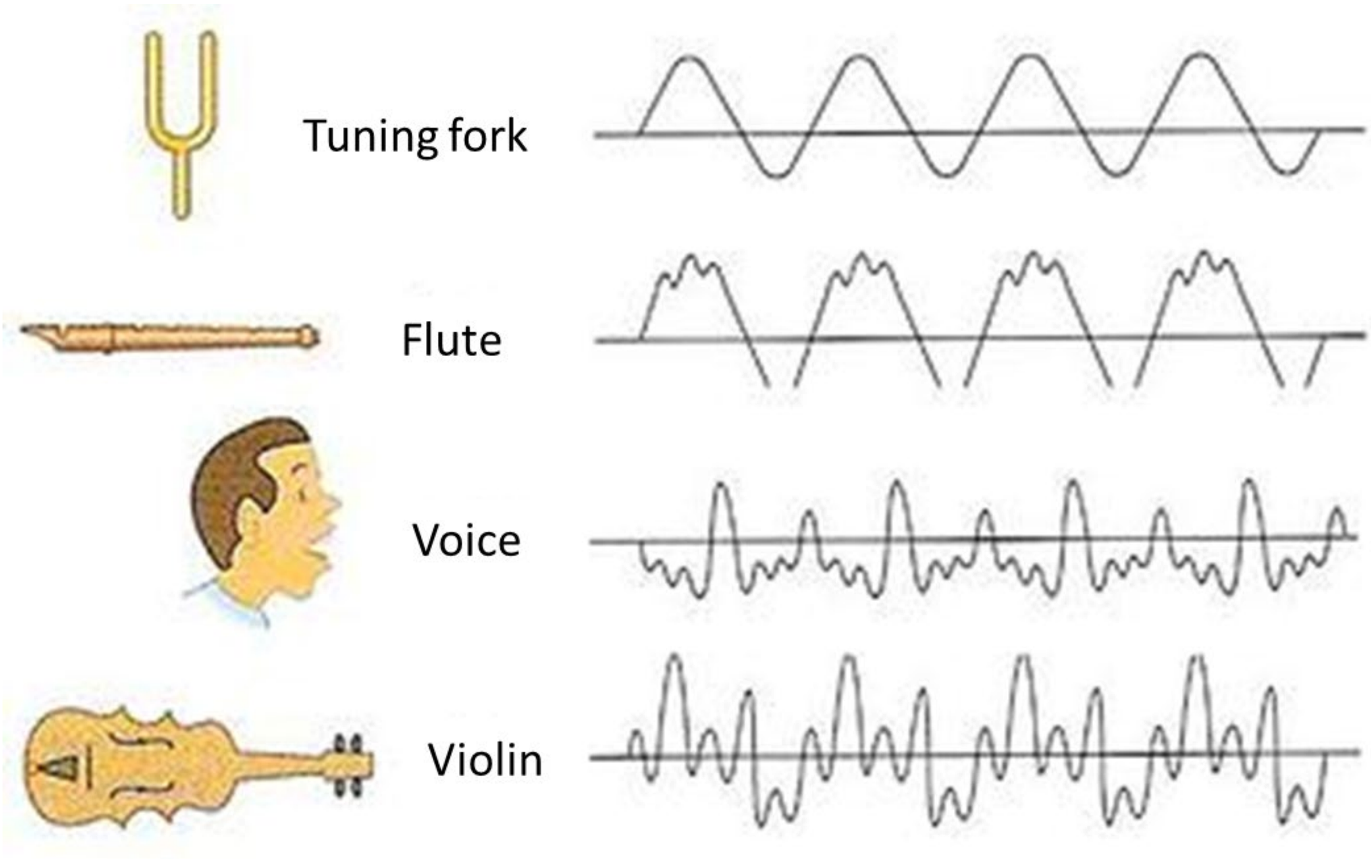
- **Sine wave:** https://www.fxsolver.com/browse/formulas/Sine+wave
- **Triangle wave:** https://www.fxsolver.com/browse/formulas/Triangle+wave+%28in+trigonometric+terms%29
- **Sawtooth Wave:** https://www.fxsolver.com/browse/formulas/Sawtooth+wave

Other Waves

https://www.youtube.com/watch?v=YsZKvLnf7wU

Other Waves

Different waves produce a different 'timbre' of sound



PASS Challege

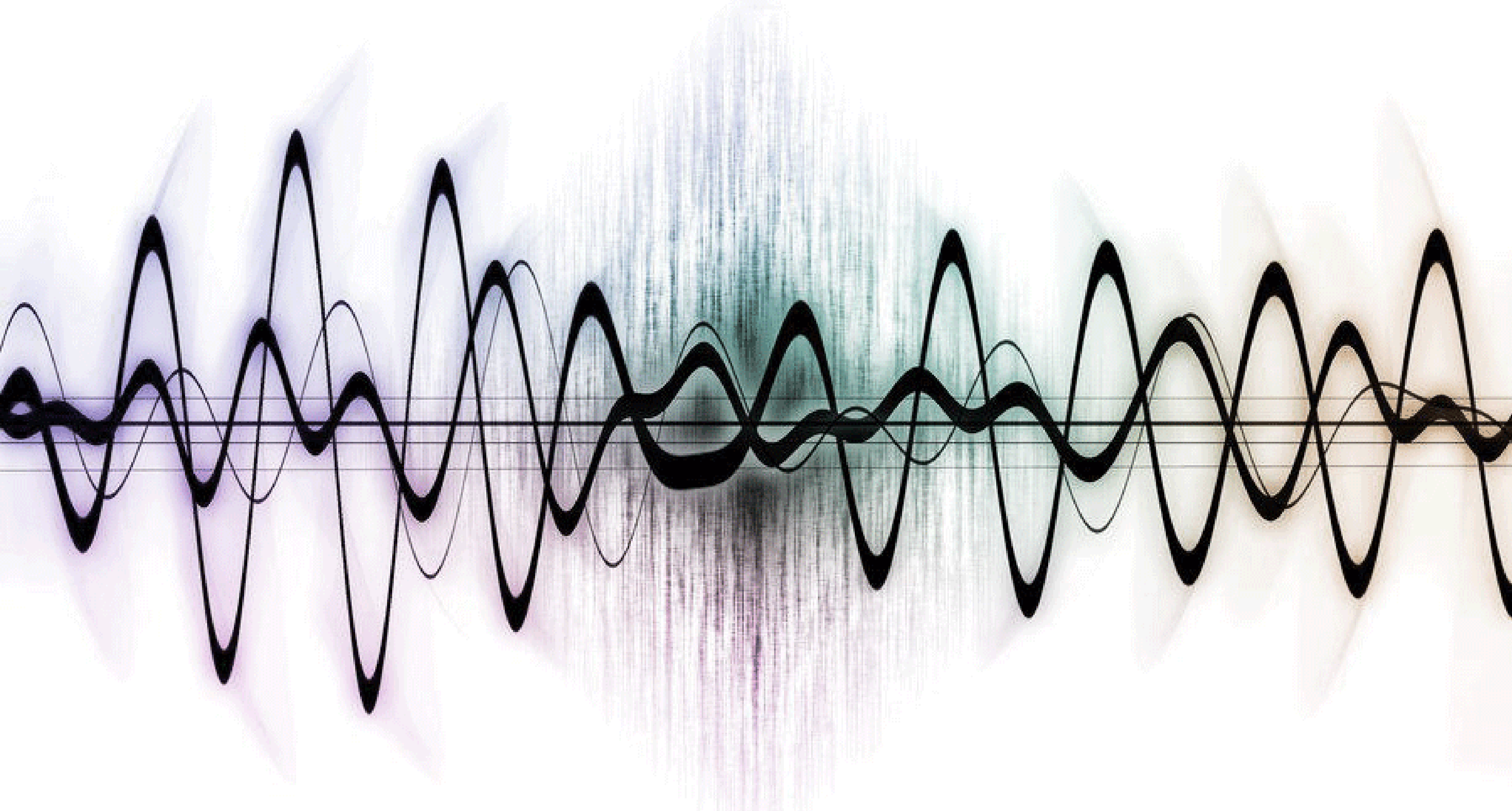
- Write a function to combine two tones from your generator together.
- Re-create a saw-tooth and/or square wave using the combination function.
- Consider impler solutions for generating square waves from a sine wave

Introduction to Digital Sound

FINAL REMARKS

Additional Resources

- **Royalty-Free Digital Sound Clips**
  http://incompetech.com/
- **Digital Sound Manipulation**
  http://www.superflashbros.net/as3sfxr/



Thank You For Listening

Michael Scott