COMP4431
Multimedia Computing

Post-Processing Effects

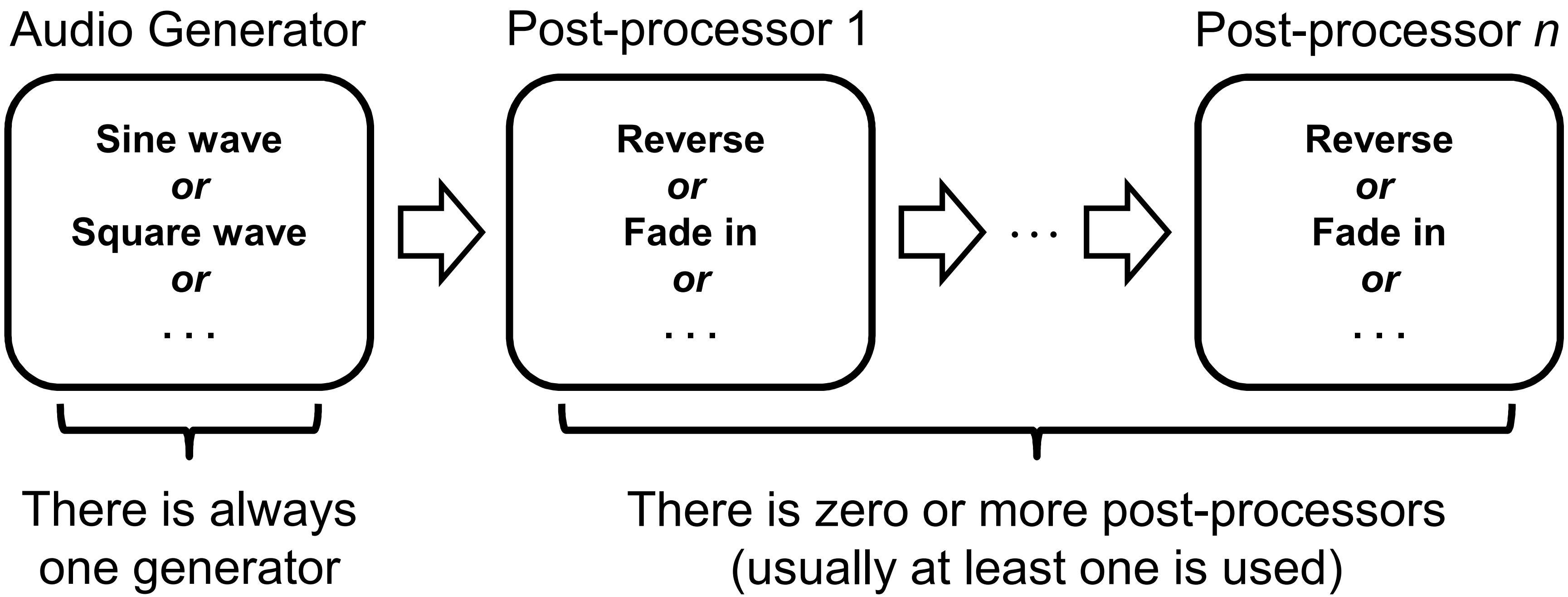
David Rossiter and Gibson Lam

This Presentation

- Post-processing effects:
  - The Basic Idea
  - Time Reversal
  - Simple Amplitude Control
  - Boost
  - Fade-In and Fade-Out
  - Tremolo
  - Exponential Decay
  - ADSR

Post-Processing

- This diagram shows the general idea of generating and 'shaping' sound



- Now we will look at some common post-processors

Time Reversal

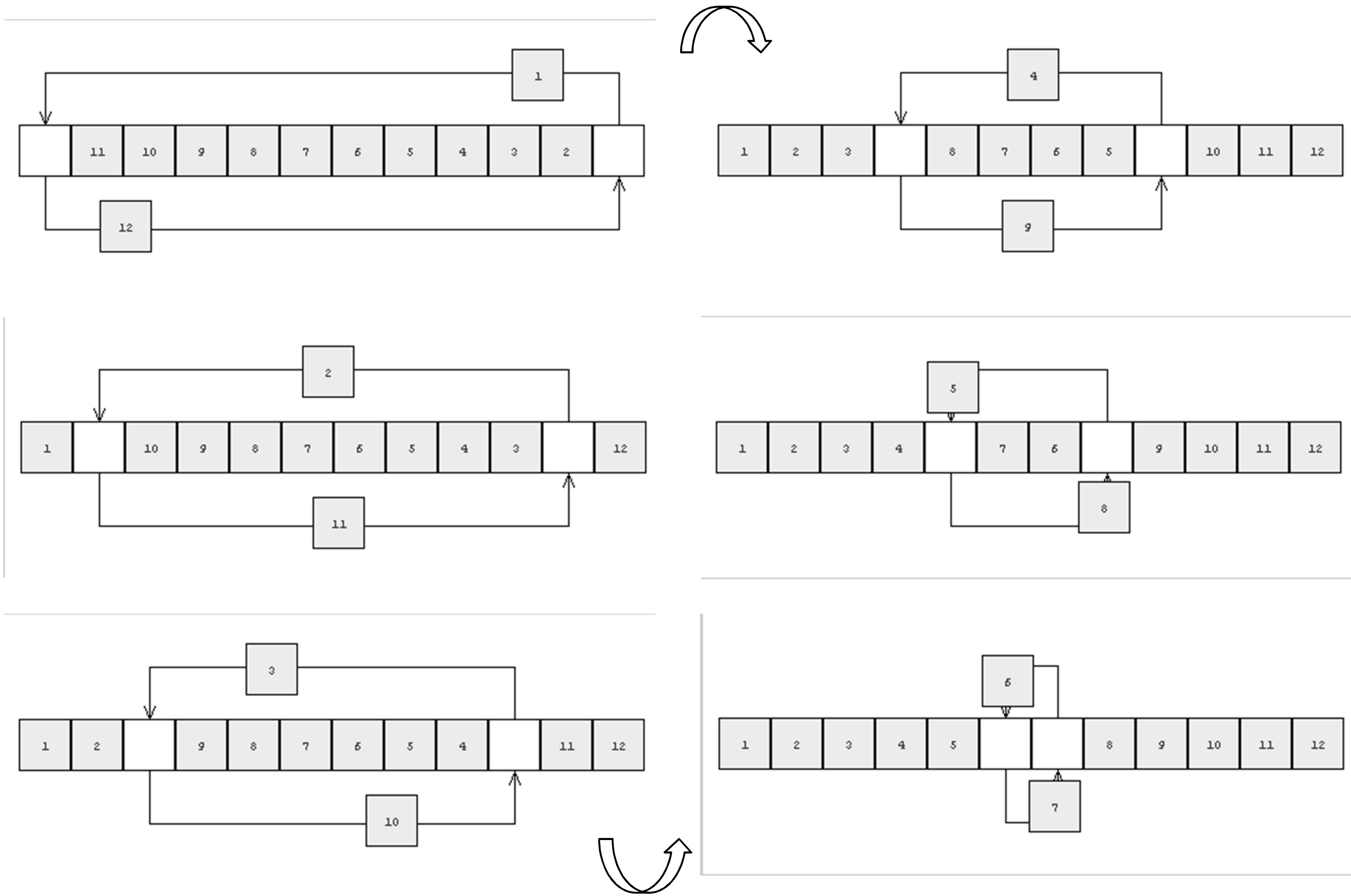
- It is easy to reverse a buffer of audio samples
- Although all the frequencies in the sound are the same as before, reversing changes how we perceive the sound (what we think about it)
- Reversing has been used as a simple method to easily generate interesting audio effects for >50 years
- A simple check whether a reversal algorithm is working properly is to reverse it again;
  the result should be the same as the original

Example Time Reversal Code

- Here is some simple code to reverse the sound samples in their time order:

```
var temp;

for (var i = 0; i < (totalSamples – 1) / 2; i++) {
    temp = samples[i];
    samples[i] = samples[totalSamples – 1 – i];
    samples[totalSamples – 1 – i] = temp;
}
```

Complete Reversal Process for 12 Samples



Simple Amplitude Control

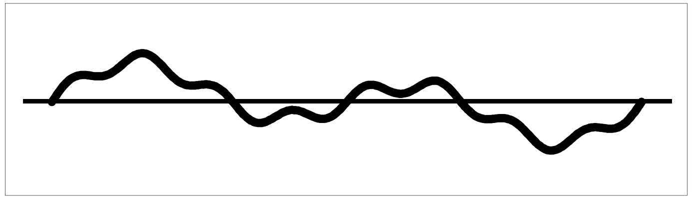
- The following code uses a variable to change the loudness of the sound:

```
var amplification = 1.2; // Make it 20% louder
for (var i = 0; i < totalSamples; i++) {
    var result = samples[i] * amplification;
    if (result > 1)
        samples[i] = 1;
    else if (result < -1)
        samples[i] = -1;
    else
        samples[i] = result;
}
```

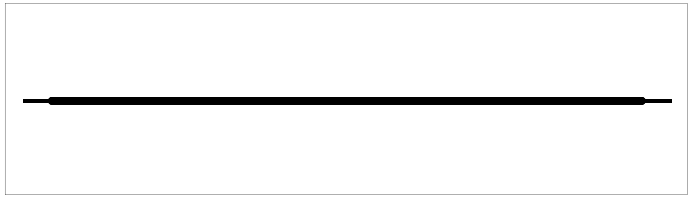
If clipping occurs, the sample will be bound by the minimum and maximum values

Volume Control

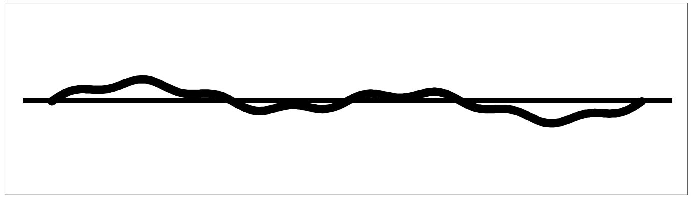
- From the previous code, it is easy to see that:
  - If `amplification` is 0, the resulting samples will become 0, which is complete silence
  - If `amplification` is smaller than 1, the resulting sound is quieter than before
  - If `amplification` is bigger than 1, the resulting sound is louder than before, so you need to be careful that clipping does not occur



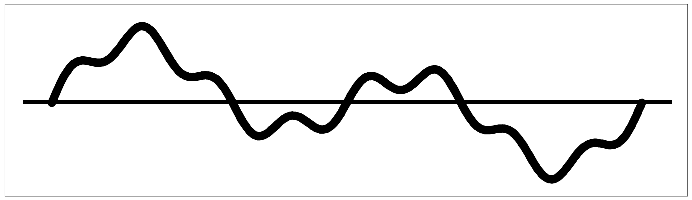
Input signal



`amplification` = 0



`amplification` < 1



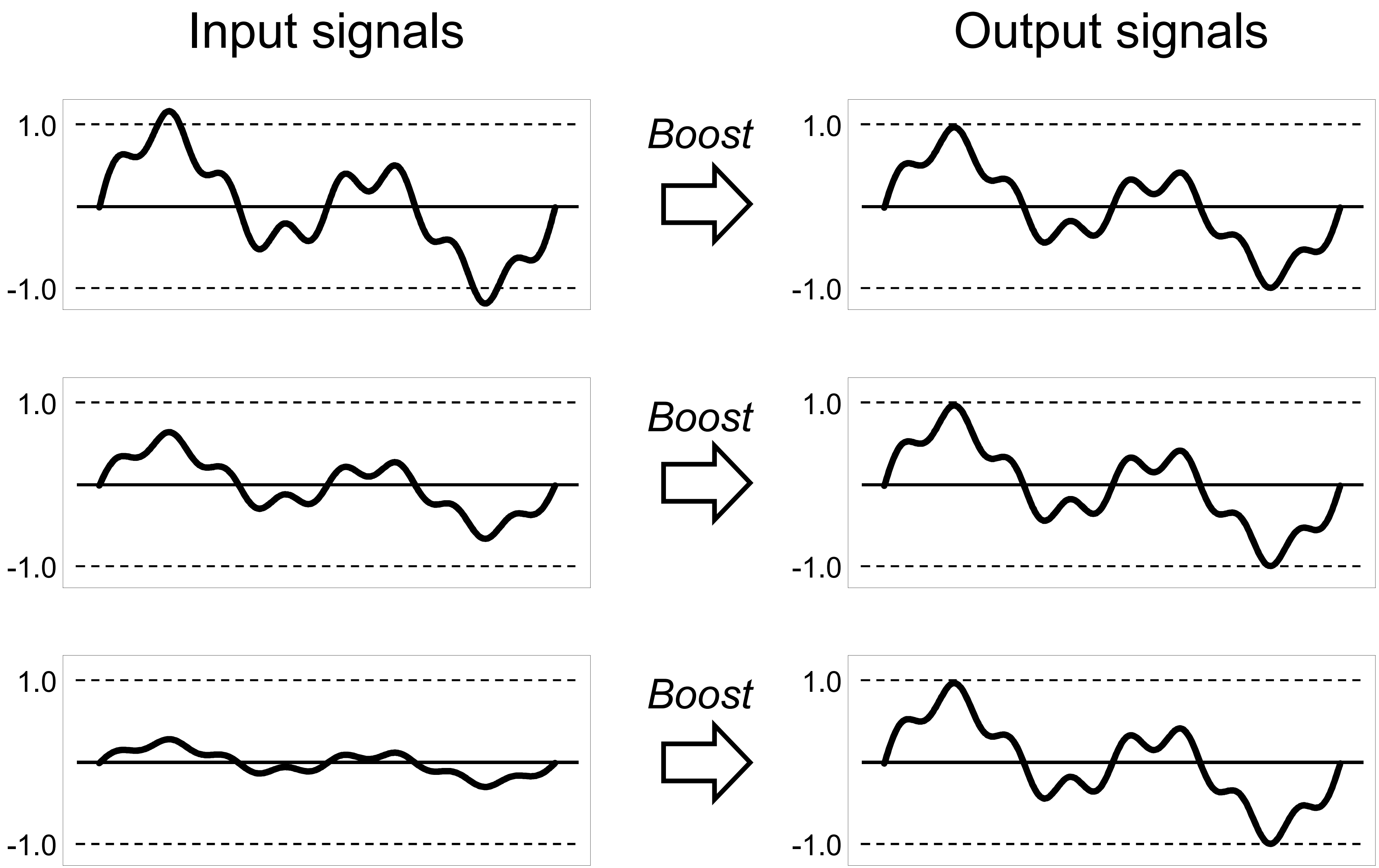
`amplification` > 1

‘Perfect’ Amplification

- Often, the user wants the general amplitude of a sound file to be as loud as possible without causing any inappropriate changes to samples
- This is hard to achieve by ‘guessing’ the appropriate multiplier value
  - It is easy to set the level too high such that the sound does become louder, but with part of the audio clipped
  - It is easy to set the level too low such that no clipping occurs but the resulting sound is not loud enough
- We can do a ‘perfect’ job using an algorithm called *boost*

Example Boost

- Assuming the maximum and minimum possible amplitudes and are 1.0 and -1.0 respectively



Boost Algorithm

1. Go through every sample
   a) Determine the highest magnitude positive sample value, call it *max*
   b) Determine the highest magnitude negative sample value, call it *min*
2. For ease of comparison, negate *min* so that it becomes positive (in other words, obtain the absolute value)
3. Compare *max* and *min*, put the largest of the two into variable *biggest*
4. Work out the multiplier which will be applied to every sample in the sound file: *multiplier = 1 / biggest*
5. Go through every sample, determine new sample value:

   *new sample value = old sample value * multiplier*

Example Boost Code

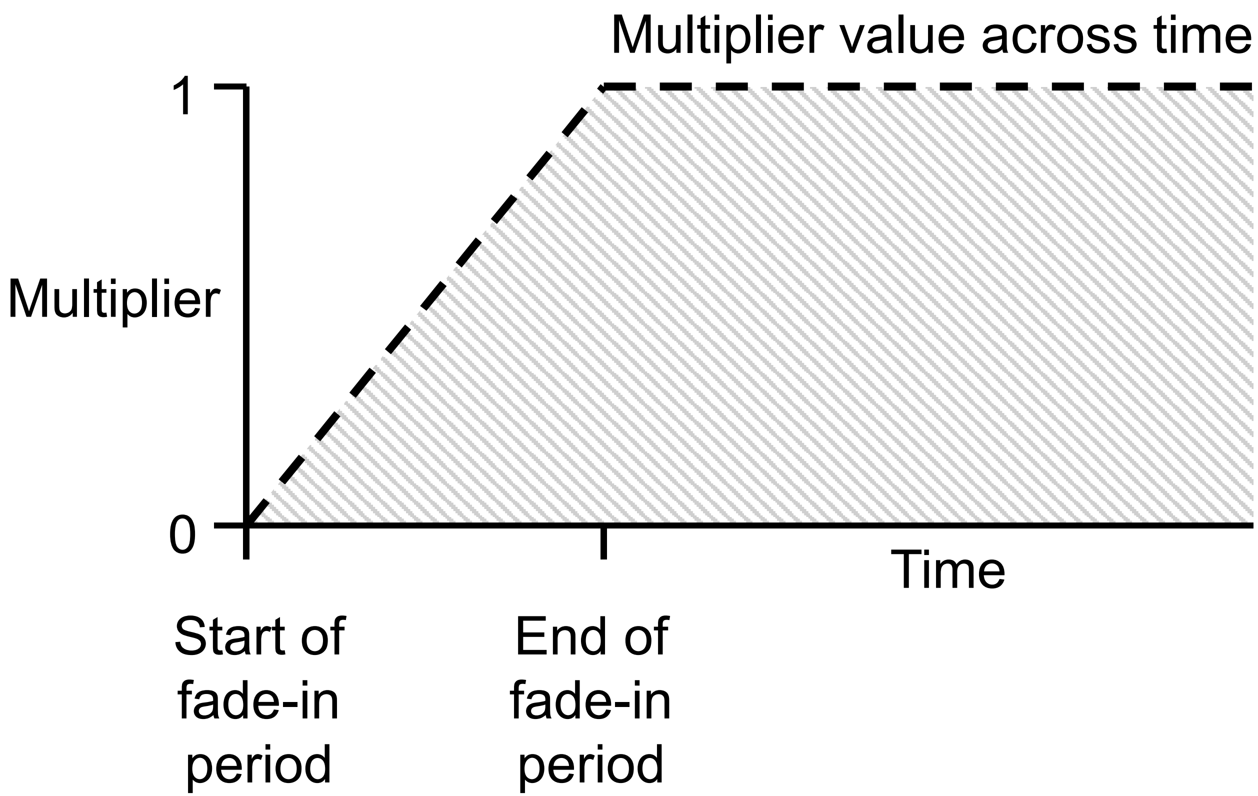
- Here is the example code:

```
var max = 0, min = 0;
for (var i = 0; i < totalSamples; i++) {
    if  (max < samples[i]) max = samples[i];
    if  (min > samples[i]) min = samples[i];
}

min = -1 * min;
var biggest = Math.max(max, min);
var multiplier = 1 / biggest;
for (var i = 0; i < totalSamples; i++) {
    samples[i] = samples[i] * multiplier;
}
```

Fade-In and Fade-Out

- It is very common that a sound starts by gradually getting louder, or quieter before it finishes
- These can be done by applying either *Fade-In* or *Fade-Out* to the audio samples
  - Fade-In means the sound starts from silence and gradually gets louder until it reaches the normal level
  - Fade-Out means the sound starts from the normal level and then gradually gets quieter until it reaches silence
- The effects apply an appropriate multiplier to the audio samples, as shown on the next slides

Fade-In Multiplier



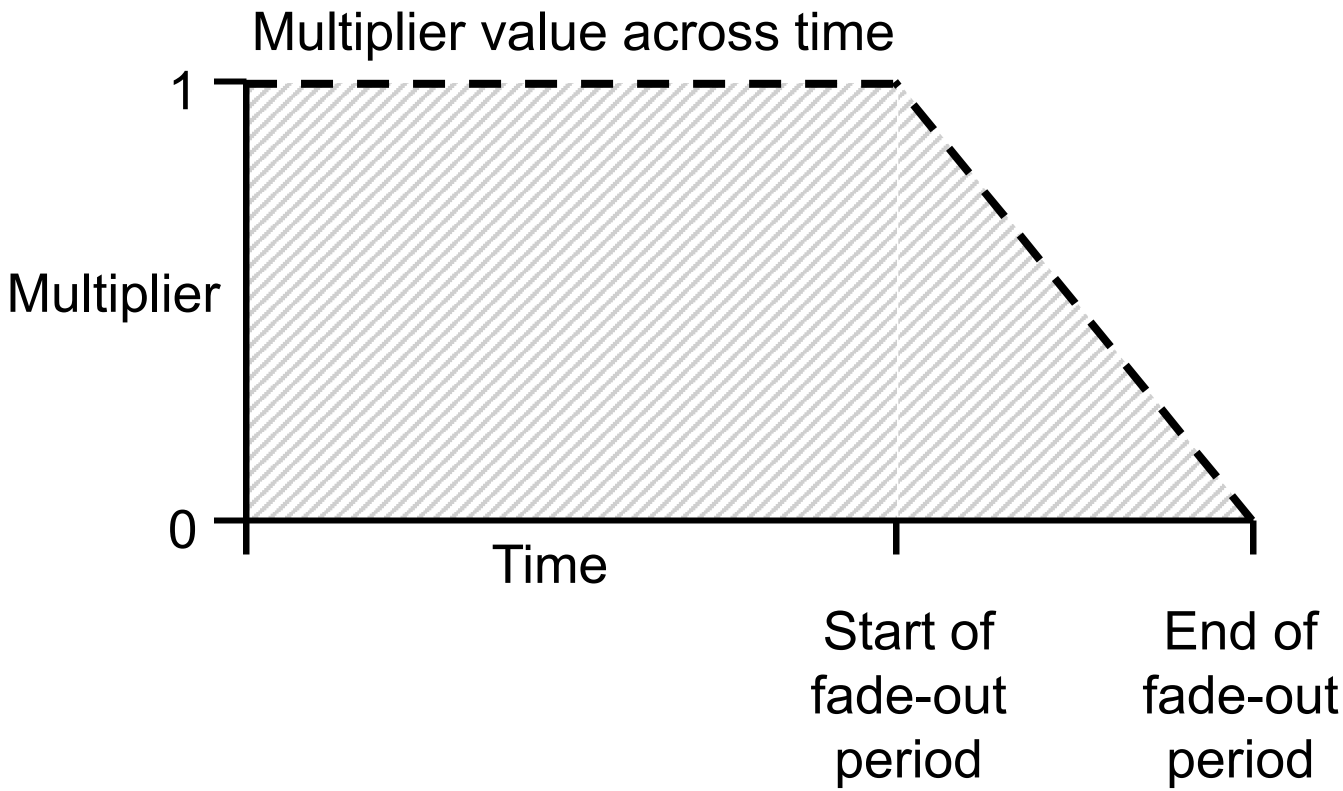
Example Fade-In Code

- The following code fades in the sound in the first two seconds:

```
var fadeDuration = 2; // The fade-in duration (secs)

var totalFadeSamples =
    parseInt(fadeDuration * sampleRate);
if (totalFadeSamples > totalSamples)
    totalFadeSamples = totalSamples;

for (var i = 0; i < totalFadeSamples; i++) {
    var multiplier = i / totalFadeSamples;
    samples[i] = samples[i] * multiplier;
}
```

Find the number of samples to change

Fade-Out Multiplier



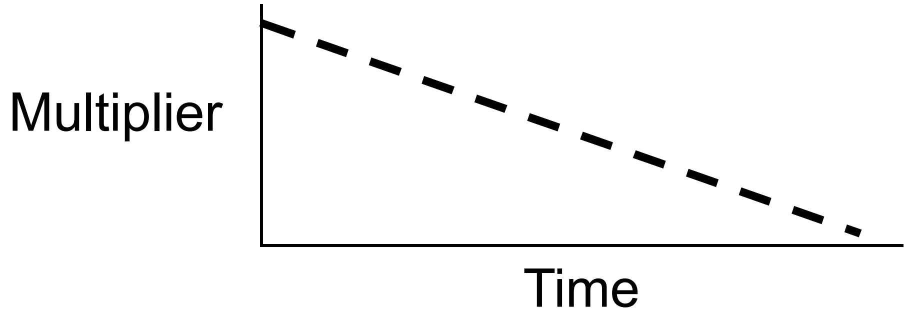
Example Fade-Out Code

- Similarly this code fades out the sound in the last two seconds:

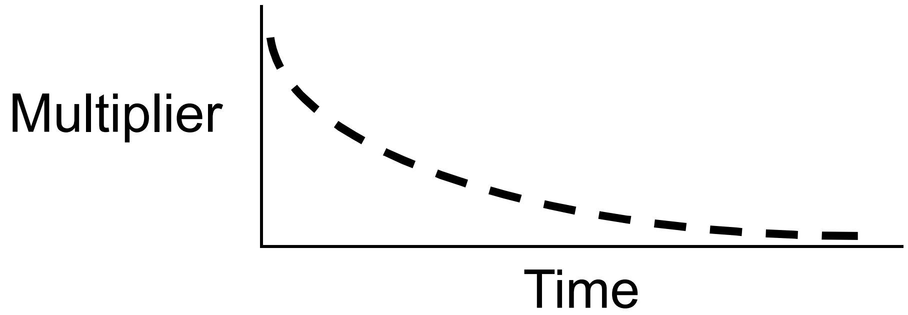
```
var fadeDuration = 2; // The fade-out duration (secs)

var totalFadeSamples =
    parseInt(fadeDuration * sampleRate);
if (totalFadeSamples > totalSamples)
    totalFadeSamples = totalSamples;

var start = totalSamples – totalFadeSamples;
for (var i = start; i < totalSamples; i++) {
    var multiplier =
        (totalSamples – 1 – i) / totalFadeSamples;
    samples[i] = samples[i] * multiplier;
}
```

Different Multipliers

- The fade-out example in the previous slides give us a linear decrease in amplitude over time:

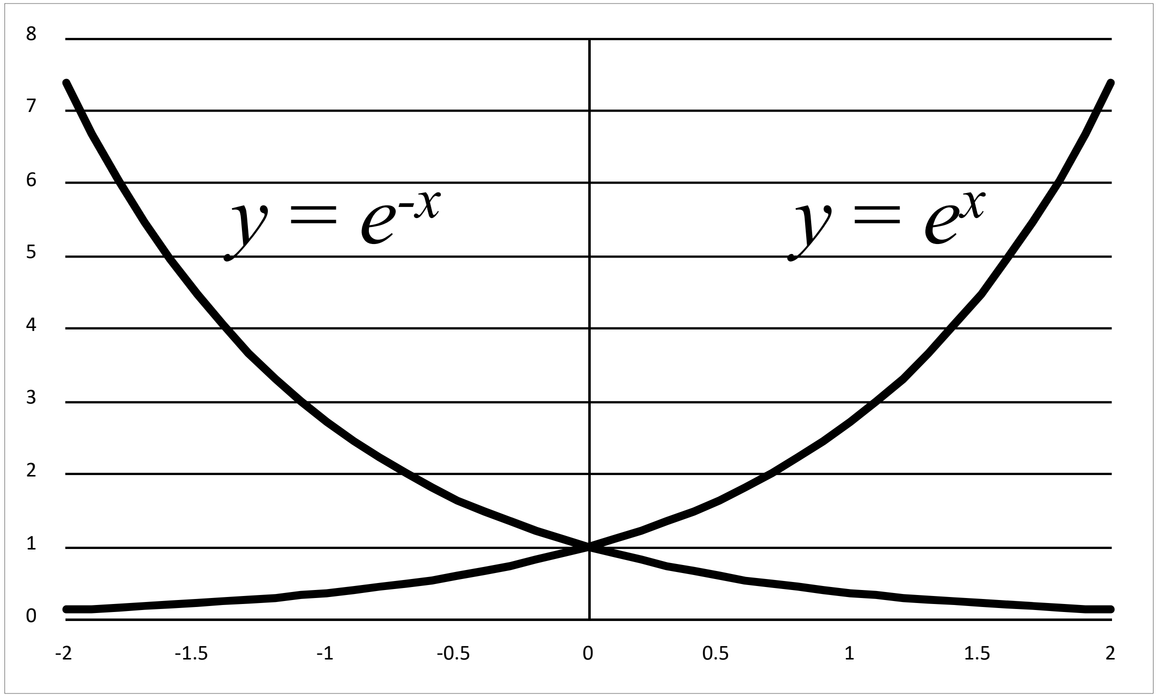


- Other multipliers can help create a more appropriate result, such as an exponential decrease:



Exponential Decay

- Exponential decay is appropriate for many sounds, including bell sounds



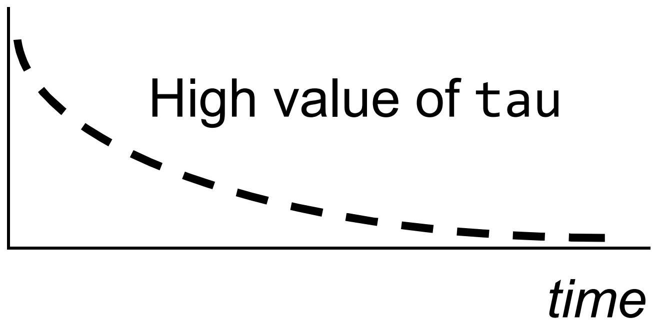
Exponential Multiplier

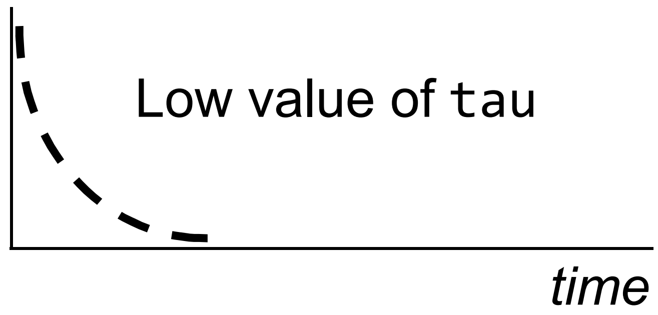
- An appropriate multiplier function is $e^{-t/\tau}$, i.e:

```
samples[i] = samples[i] * Math.exp(-t / tau);
```

where

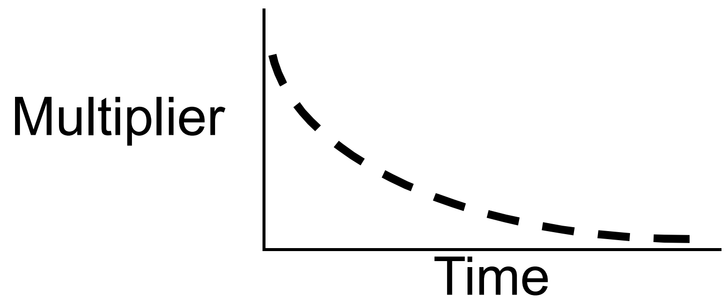
- `t` is the current time
- `tau` gives us a simple way to control the rate at which the sound gets quieter, i.e.:





Example Exponential Decay Code

- Here is some simple code to apply exponential decay to the entire audio:

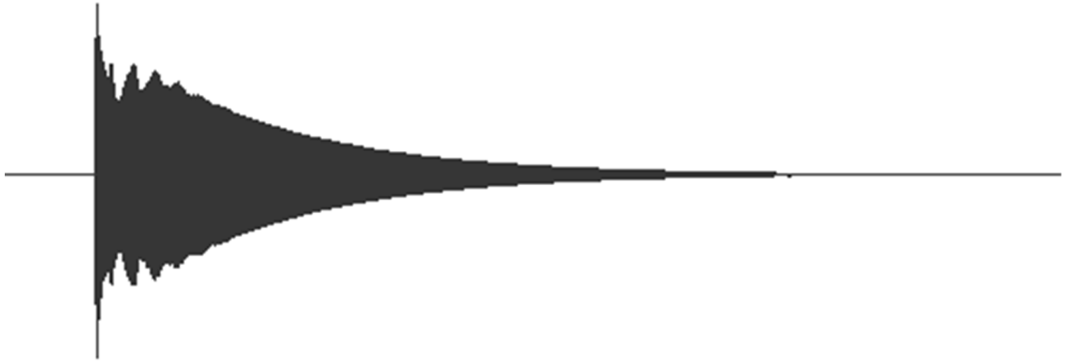



```
var timeConstant = 0.2;

for (var i = 0; i < totalSamples; i++) {
    var t = i / sampleRate;
    var multiplier = Math.exp(-t / timeConstant);
    samples[i] = samples[i] * multiplier;
}
```

Tremolo

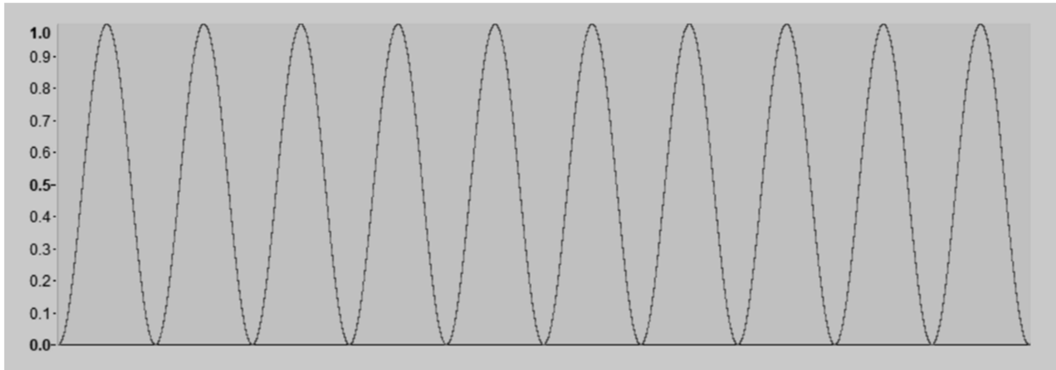
- The basic idea of tremolo is to multiply the sound with a sine wave ranging in amplitude from 0 to 1
- This can produce a range of different results, depending on how it is used
- If the tremolo multiplier frequency is low e.g. <20-30Hz then the user can easily hear the sound get louder and quieter in a pleasant pattern

Input Sound



7s

Multiplier

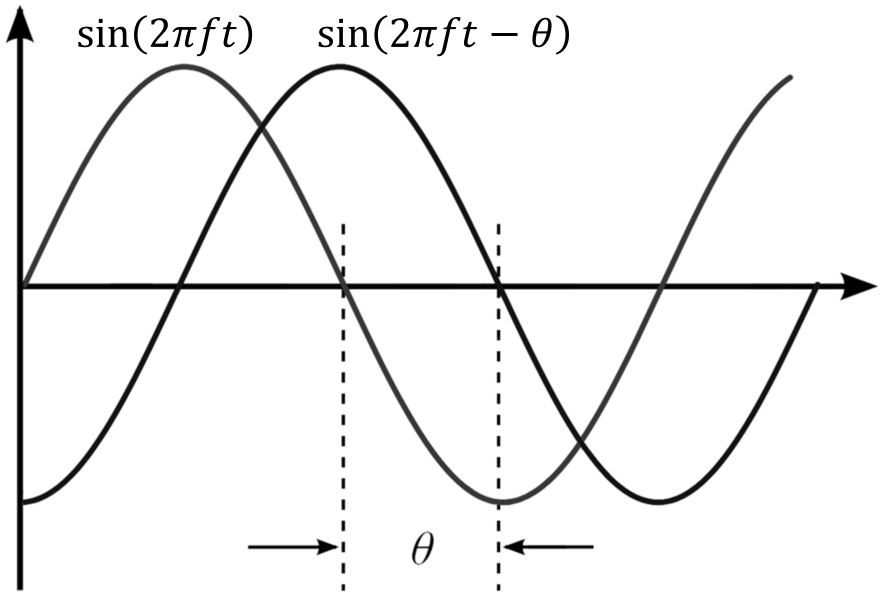


Result

7s

The Multiplier

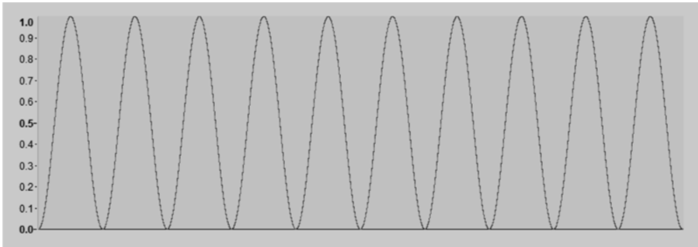
- The multiplier usually starts from 0
- That means the sine function has to be adjusted appropriately for the starting phase



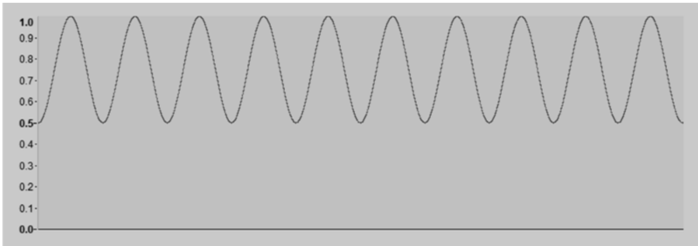
- The amplitude of the sine wave also has to be normalised within 0 and 1

Multiplier Wetness

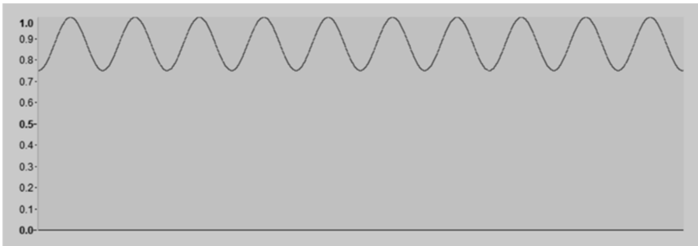
- For maximum effect, the multiplier is in the range from 0 to 1
- However, the lower value (0) can be raised
- The higher it is raised, the less obvious the effect
- This parameter is called ‘wetness’



Wetness = 1.0



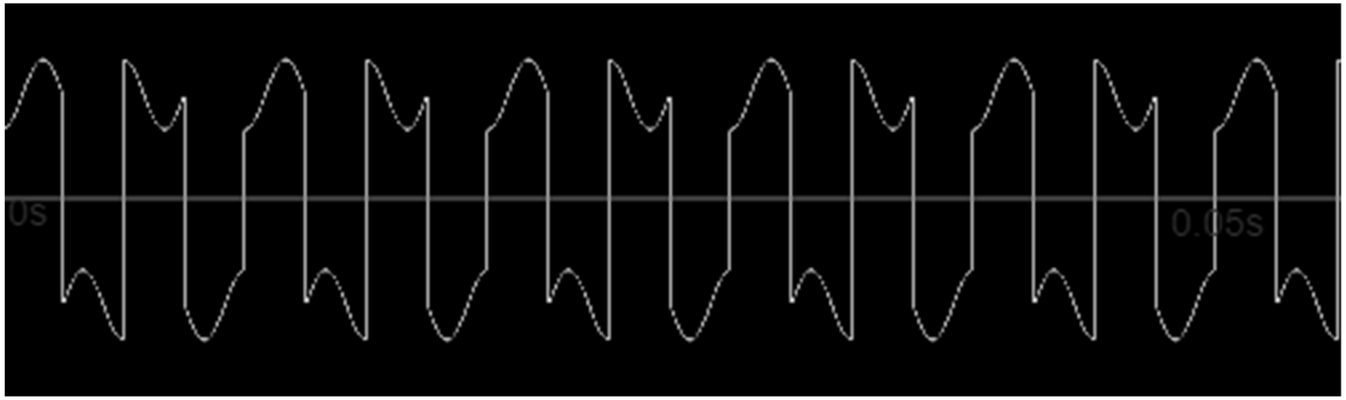
Wetness = 0.5



Wetness = 0.25

Higher Tremolo Frequencies

- However, if the tremolo frequency is high, e.g. hundreds of Hz, then a new waveform can be created

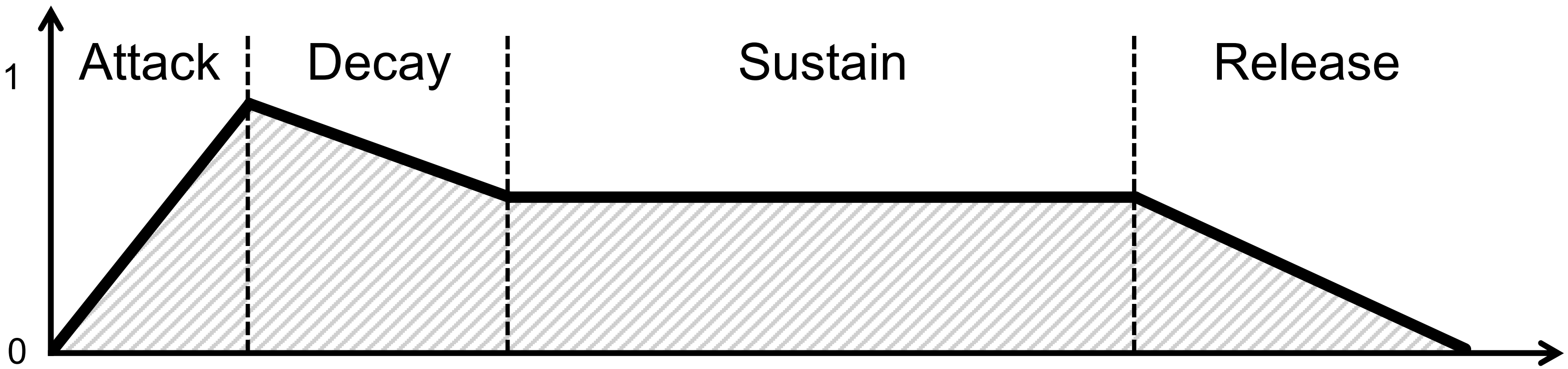


- The above image shows a 200Hz square wave with 300Hz tremolo applied (wetness = 0.5)

The ADSR Envelope

- The multipliers used in the previous effects adjust the amplitude of a sound, which are generally called the envelopes of the sound
- In sound synthesizers, a more complete and commonly used envelope is called the ADSR envelope
- Each letter of ‘ADSR’ represents a separate stage:
  - A for Attack, D for Decay, S for Sustain and R for Release

Stages in the ADSR Envelope



Attack

- When a sound is started, e.g. pressing on a piano key, the volume goes from 0 to the maximum volume

Decay

- After the attack, the volume drops gradually to a steady level

Sustain

- This is the steady level of the sound which stays for the majority duration of the sound

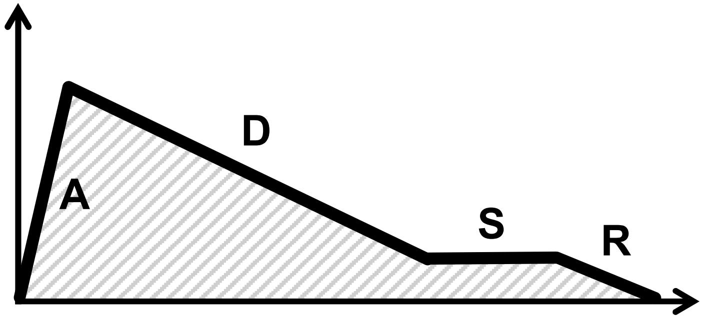
Release

- When a sound is stopped, e.g. releasing the piano key, the volume drops off to 0 gradually

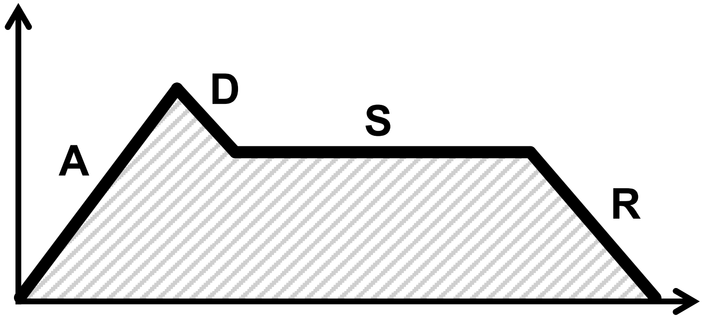
Parameters for ADSR

- There are five parameters for the ADSR envelope:
  - Attack time
    - The time the sound reaches the maximum amplitude
  - Decay time
    - The time the sound drops to the sustain level
  - Sustain level
    - The level of the steady state, e.g. as a percentage of the maximum amplitude
  - Release time
    - The time for the sound to completely die out
  - Duration
    - The duration of the sound

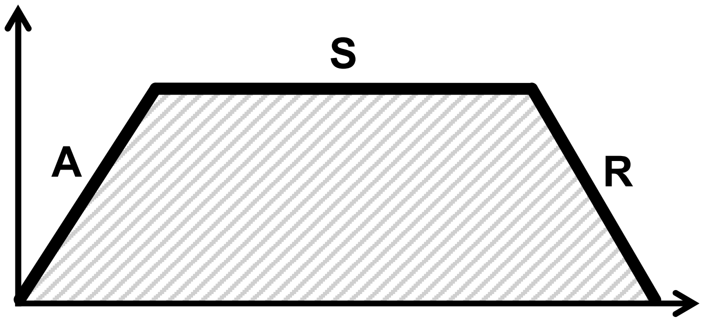
Some Example ADSR Envelopes



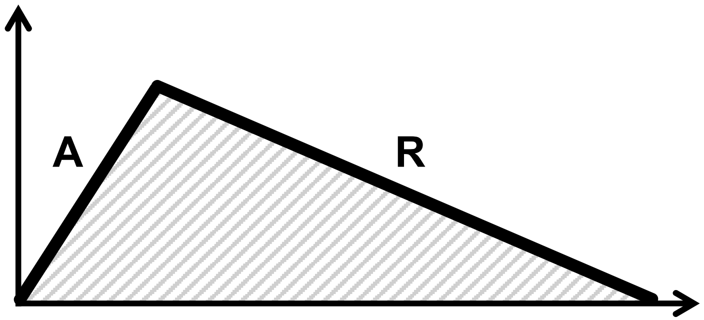
Example Piano Envelope



Example Trumpet Envelope



Example Violin Envelope



Example Guitar Envelope