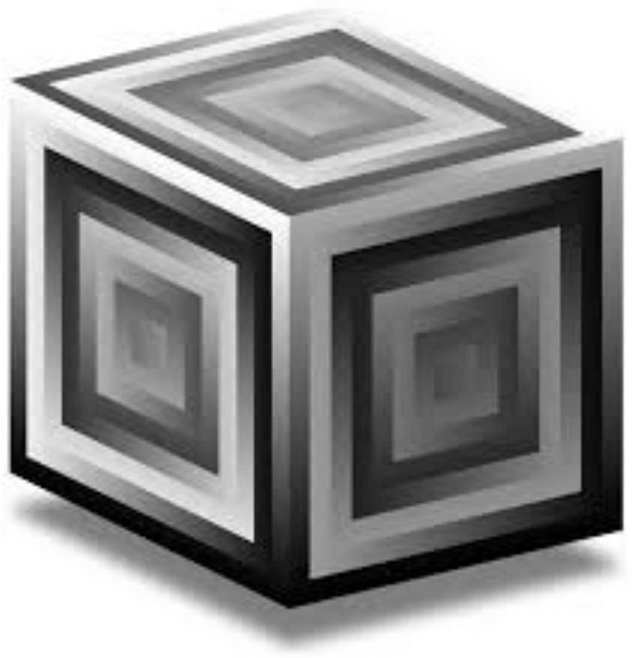
Envelopes

Topics Addressed

- Env/EnvGen
- doneActions
- ASR/ADSR
    - Gates
- Triggering ADSR

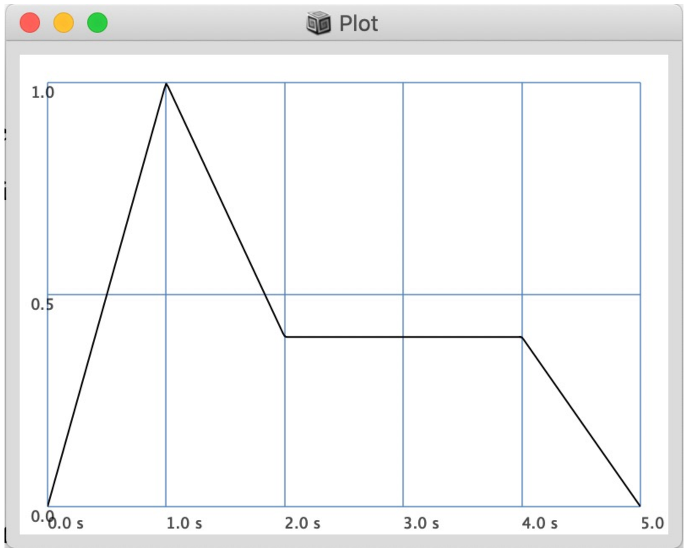


Amplitude Through Time

- An important characteristic of sound is its volume as it passes through time. We can this the envelope of a sound.
- Most sounds, like those from instruments, can be subdivided into three parts based on their envelope: attack, sustain, release
  - The attack in particular plays an important part in discerning what created the sound. For example, removing the attack from a flute’s note will produce an almost electronic sound.
- SuperCollider provides several UGens that can be used to shape a sound’s amplitude through sound.

Env

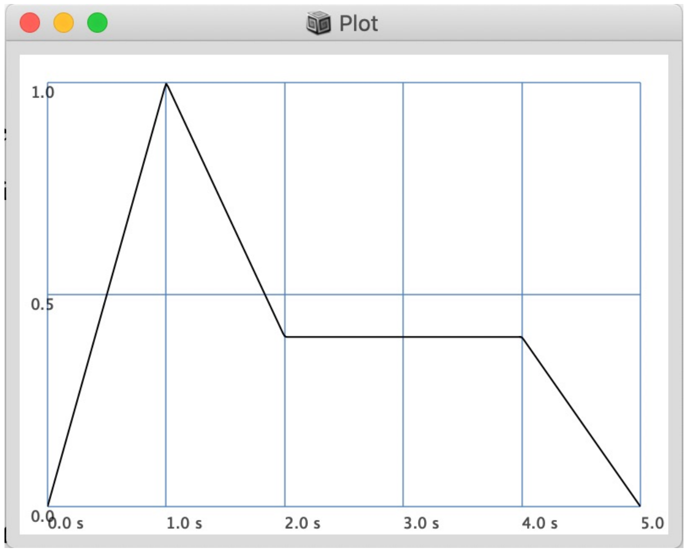
- Env, short for envelope, defines a graph with amplitude levels and durations in time segments.



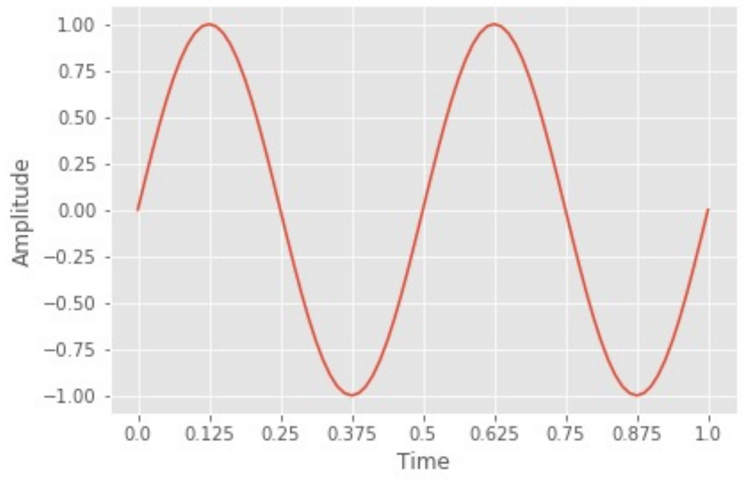
```
(
Env.new(
    [0, 1, 0.4, 0.4, 0],
    [1, 1, 2, 1],
    curve: 'lin'
).plot;
)
```

1st argument to Env is levels (i.e., amplitudes) and the 2nd argument to Env is times in seconds to each amplitude.

Combining Signals with Envelopes



?

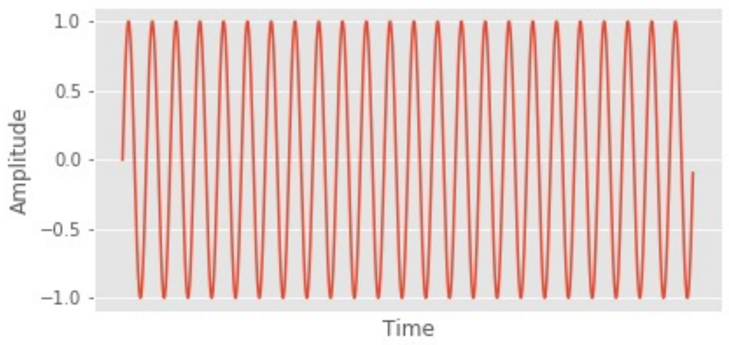


How can we combine an envelope with a signal mathematically?

MULTIPLICATION

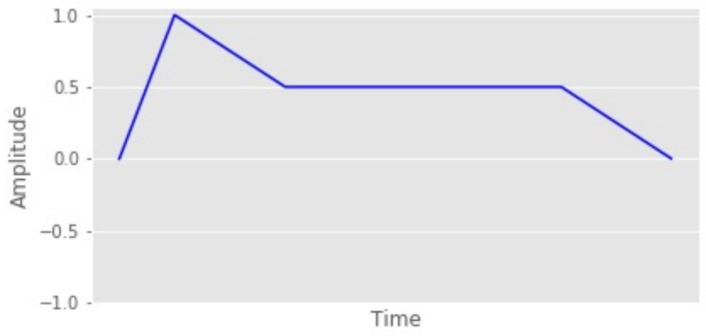
Slide showing combined waves

Signal

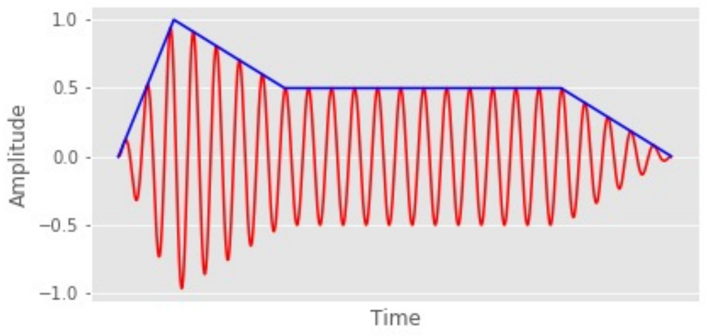


*

Envelope



=



This is y-value by y-value multiplication. The peak amplitude of the signal is capped by the envelope.

EnvGen

- The class Env defines an envelope in much the same way that a SynthDef defines a synth. To create an instance of the envelope on the Server, we use EnvGen.
- Like all UGens, EnvGen comes with class methods `.ar` and `.kr`. When using EnvGen to modulate the amplitude, we should use `.kr`.
  - .ar is primarily used for audio signals (i.e., the actual samples)
  - .kr is primarily used to modulate audio signals
- The class EnvGen is primarily used in conjunction with the class Env.

Simple Example

```
s.plotTree;

SynthDef(\sineEnv, {
    arg freq = 440;
    var env, sig;
    env = Env.new(
        [0, 1, 0.4, 0.4, 0],
        [1, 1, 2, 1],
        curve: 'lin'
    );
    env = EnvGen.kr(env);
    Out.ar(0, SinOsc.ar(freq) * env ! 2);
}).add;

~s1 = Synth(\sineEnv);
~s1.free;
```

- Note that we are using `EnvGen.kr` because we will modulate the signal (i.e., the sine wave) with this object.
- Observe the plot tree when executing the synth. It does not remove itself from the server after the envelope is complete. Imagine a piece that played thousands or tens of thousands of notes. These synths use up valuable computer resources. We need to remove synths after they are done executing.
  - One option is to free the synth but this requires the user to manually execute free commands.
  - A better option is to use doneActions.

doneActions

- SC’s EnvGen and several other UGens have an optional parameter called doneAction which specifies to the server what should happen after the envelope completes.
- doneActions are incredibly useful. Documentation for them can be found under the Done class, providing 16 class methods that control Synth behavior
  - The most useful is Done.freeSelf which removes the Synth from the Node tree on the server

```
EnvGen.kr(Env([0, 1, 0], [1, 1]), doneAction: Done.freeSelf)
```

doneActions

name	value	description
none	0	do nothing when the UGen is finished
pauseSelf	1	pause the enclosing synth, but do not free it
freeSelf	2	free the enclosing synth
freeSelfAndPrev	3	free both this synth and the preceding node
freeSelfAndNext	4	free both this synth and the following node
freeSelfAndFreeAllInPrev	5	free this synth; if the preceding node is a group then do g_freeAll on it, else free it
freeSelfAndFreeAllInNext	6	free this synth; if the following node is a group then do g_freeAll on it, else free it
freeSelfToHead	7	free this synth and all preceding nodes in this group
freeSelfToTail	8	free this synth and all following nodes in this group
freeSelfPausePrev	9	free this synth and pause the preceding node
freeSelfPauseNext	10	free this synth and pause the following node
freeSelfAndDeepFreePrev	11	free this synth and if the preceding node is a group then do g_deepFree on it, else free it
freeSelfAndDeepFreeNext	12	free this synth and if the following node is a group then do g_deepFree on it, else free it
freeAllInGroup	13	free this synth and all other nodes in this group (before and after)
freeGroup	14	free the enclosing group and all nodes within it (including this synth)
freeSelfResumeNext	15	free this synth and resume the following node

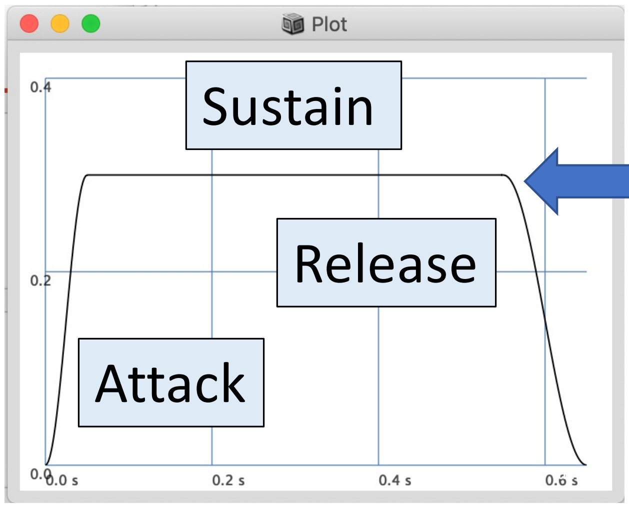
Simple Example with doneAction

```
s.plotTree;

SynthDef(\sineEnvDone, {
    arg freq = 440;
    var env, sig;
    env = Env.new(
        [0, 1, 0.4, 0.4, 0],
        [1, 1, 2, 1],
        curve: 'lin'
    );
    env = EnvGen.kr(env, doneAction: 2);
    Out.ar(0, SinOsc.ar(freq) * env ! 2);
}).add;

~s1 = Synth(\sineEnvDone);
```

- Here we change the previous example to include a `doneAction` of 2 which frees the node after the envelope is complete
- Note that we don't need a specific free command. The EnvGen UGen handles that task for us.
- doneActions are wonderful and important components that remove the user from manually handling the job of freeing nodes on the server.

ASR

We can form a common envelop pattern called ASR (Attack Sustain Release) using the class method `.linen`. Note that the sustain is a *fixed* time, though it can be parameterized.



Note that all envelopes accept a curvature which need not be a line between points. See documentation for different types of curvature.

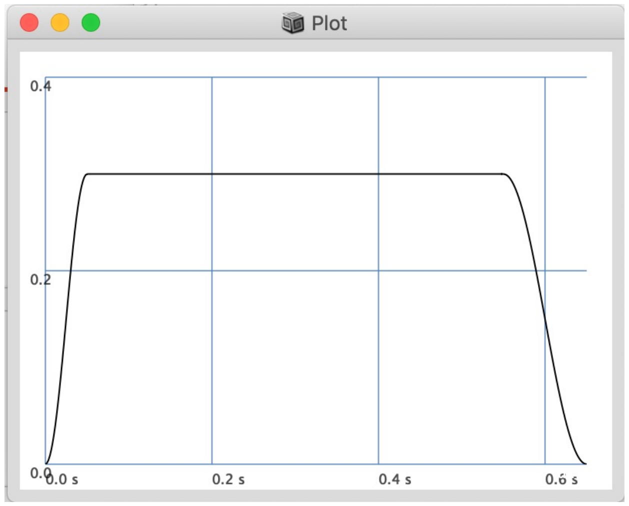
Attack Sustain Release Level

```
env = Env.linen(0.05, 0.5, 0.1, 0.3, 'sine');
```

ASR Example

```
(
SynthDef(\shortSine, {
        arg freq = 440;
        var env, sig;
        env = Env.linen(0.05, 0.5, 0.1, 0.3, 'sine');
        env = EnvGen.kr(env, doneAction: 2);
        sig = SinOsc.ar(freq);
        Out.ar(0, sig * env ! 2);
}).add;
)
```

Envelope for the Env on the left



EnvGen and Gate

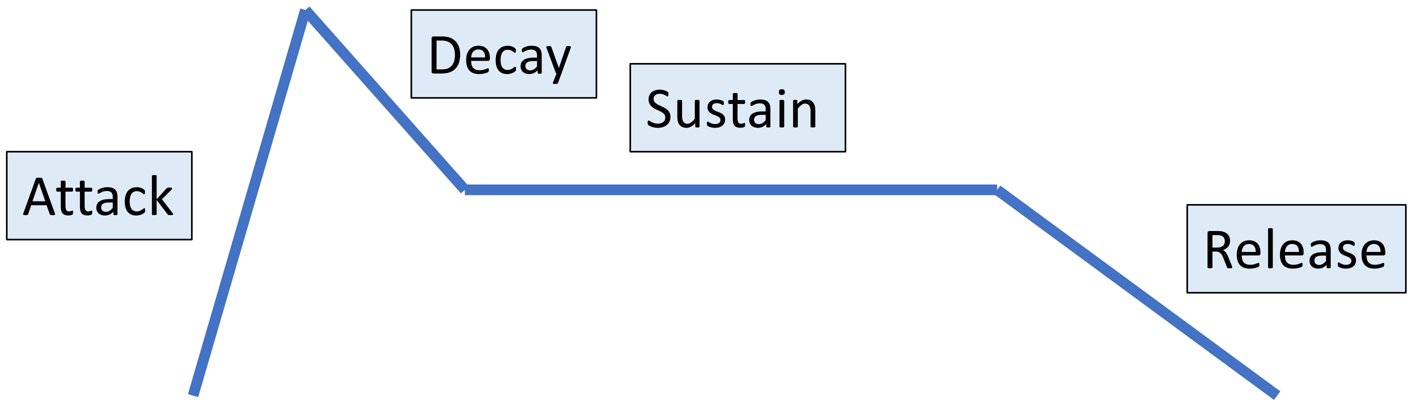
```
(
SynthDef(\gatedSine, {
	arg freq = 440, gate = 0;
	var env, sig;
	env = Env.linen(0.05, 1, 0.1, 0.3, 'sine');
	env = EnvGen.kr(env, gate);
	sig = SinOsc.ar(freq);
	Out.ar(0, sig * env ! 2);
}).add;
)

~s1 = Synth(\gatedSine);
~s1.set(\gate, 1);
~s1.set(\gate, 0);
~s1.free;
```

- The class EnvGen can also accept an additional parameter called a gate which can be used to trigger the envelope.
- The gate parameter is a powerful tool that we can use to retrigger the same synth.
- Note here that the Synth is never removed from the server with a doneAction. When the Envelope finishes the amplitude is zero but it it is still ”playing”.
- Note that the gate needs to be reset to zero before being re-triggered with a gate of 1
- Gates become more complicated for envelopes that do **not** have a fixed sustain duration. Here though, the .linen class method has a fixed sustain so a simple trigger of 1 suffices.

\shortSawPattern

- See .scd file for this lecture for the code
- Important notes
  - The class `Impulse` creates a sample of amplitude one followed by samples of zero at a regular frequency. Here we use this to trigger the gate of an EnvGen at a regular rate.
  - The class `Select` is a way of choosing between several options on the server. It is the closest we can get conditional structure on the server side. Here the Select chooses from randomly generated integers from the `TIRand` class several possible frequencies to pass to the `Saw` oscillator.
  - Given some initial frequency which we can consider the tonic, the four possible frequencies are scale degrees $\hat{1}$, $\flat\hat{3}$, $\hat{5}$, and $\flat\hat{6}$. These are randomized by octave.

ADSR



- One of the most common envelope generators is ADSR: Attack, Decay, Sustain, Release
- Concept developed by Vladimir Ussachevsky at Princeton in 1965 while working on the Moog synthesizer.
- Universal on pretty much all digital/analog synthesizers

ADSR Triggering

- The class method `Env.adsr` produces an ADSR envelop. Note though that it is **not** a fixed time for sustain.
- To trigger the release of an ADSR, zero must be passed to the gate parameter otherwise the Synth will persist at the sustain portion of the envelope.
- This is incredibly useful for input devices like keyboards, for example, where the down stroke of the key can be mapped to a gate of 1 and the release can be mapped to 0. We will see more examples of this when we discuss MIDI.

ADSR Example

```
(
SynthDef(\tri, {
	arg freq = 440, aTime = 0.1, dTime = 2.5, rTime = 1.5, peakAmp = 0.4, sAmp = 0.3, gate = 1;
	var sig, env;
	sig = LFTri.ar(freq);
	env = Env.adsr(aTime, dTime, sAmp, rTime, peakAmp);
	Out.ar(0, sig * EnvGen.kr(env, gate, doneAction: Done.freeSelf) ! 2);
}).add;
);

a = Synth(\tri, [\freq, 150]);
a.set(\gate, 0);
```

- Instantiating the Synth will trigger the envelope because the default value for the gate is set to 1. This will play the attack and decay and hold on the sustain. Note there is **no** sustain time provided. It plays as long as the gate is open.
- To release the note we must send the Synth a message to set the gate to 0.
- Note that we cannot reuse this Synth because we set the doneAction to free the Synth upon completion of the envelope.