SuperCollider Tutorial

Chapter 6

By Celeste Hutchins

2005

www.celesteh.com

Additive Synthesis

Additive synthesis is the addition of sine tones, usually in a harmonic series, to create complicated tones. We've seen additive synthesis before. The last example of chapter 2, where we played the first four overtones of 100 Hz was an example of additive synthesis. As was the project associated with the previous chapter.

There are some UGens that are helpful for adding sine tones. You can mix multiple signals by adding them all together and dividing by the number of signals. For example:

```
    result = (sine1 + sine2 + sine3 + sine4 + sine5) / 5;
```

You can also use a UGen called Mix which takes an array of signals and, appropriately enough, mixes them together. It's like Pan in reverse.

```
     result = Mix.ar([sine1, sine2, sine3, sine4, sine5]);
```

Additive synthesis uses sine tones because sine tones, like those produced by SinOsc, are pure. They have no overtones. They are only their frequency and have no other pitch content. Any other, more complicated tone that you hear is actually a collection of sine tones. When you hear a tone with two overtones, you are hearing a sine wave of the fundamental and the sine waves of the two overtones. Periodic waves such as triangle, pulse and sawtooth can all be describes in terms of what sine tones comprise them. We'll come back to this later.

Complicated sounds in the real world are made up of many, many sine tones. Theoretically, we can synthesize any tone from a combination of component sine tones. In practice, this may be too complicated for some noisy tones. Additive synthesis is used in organs. The addition of “stops,” or pipes, is the addition of extra tones to create a complex timbre. Electric organs, like the Hammond B3 also use additive synthesis.

Ring Modulation

On analog synthesizer, like the ones Nicole has been collecting, every oscillator has it’s own circuit. Doing additive synthesis would have required a bank of tunable oscillators. That would have gotten expensive. Pipe organs don’t have a way of varying what comes out of a pipe. But synthesizer designers invented ways for composers to **modulate** their oscillators. One early method of modulation is called **ring modulation**.

Ring Modulation got it's name because when people first started building hardware to do this, a bunch of components were soldiered around in a circle, so it was a "ring" modulator. The result is a noisy sound. Ring Modulators work by multiplying the carrier signal times the modulator signal. RM is expressed arithmetically with multiplication: carrier * modulator. Note that these are the waveforms that we’re multiplying and not the frequencies.

```
modulator = SinOsc.ar(modulator_freq, mul: modulator_amp);
carrier = SinOsc.ar(carrier_freq, mul: carrier_amp);
result = modulator * carrier;
```

Remember, though, that we have a mul argument to SinOsc

```
modulator = SinOsc.ar(modulator_freq, mul: modulator_amp);
result = SinOsc.ar(carrier_freq, mul: modulator * carrier_amp);
```

That creates the exact same result. Because the carrier (aka, the result) is the last oscillator in a chain, the variable controlling its amplitude is usually simply called “amp”.

If we write this into a display and use predefined ControlValues for the modulator_freq and the carrier_freq, both of them are only in the audio range. We want to be able to use the modulator as a Low Frequency Oscillator (sometimes called an **LFO**) to modulate the carrier with waves that are too slow to hear. The ControlValue helpfile mentions another class called **ControlSpec**. According to its own helpfile, a ControlSpec is a, “specification for a control input.” An instance of ControlSpec is an object that describes the range of an instance of a ControlValue. The spec_ setter for ControlValues can either take a symbol that refers to a predefined ControlSpec, or it can take a ControlSpec defined by the programmer. The ControlSpec constructor is described as

```
    ControlSpec.new( minval, maxval, warp, step, default,units);
```

We want a range from close to 0 to 20,00 Hz. Since our perception of tone is exponential (each octave is double of the Hz of the octave below), she wants the slider to increase exponentially, with a small a step value as possible.

```
    modulator_freq.spec_(ControlSpec(0.1, 20000, \exp, 0, 1));
```

Our Display for Ring Modulation is below.

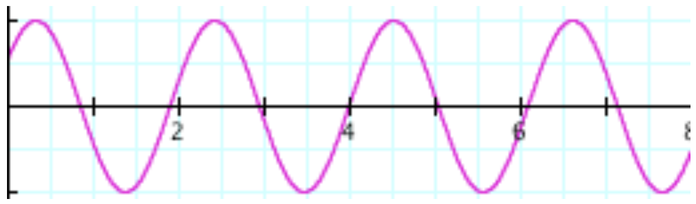
```
(

    Display.make({ arg thisDisplay, carrier_freq, modulator_freq,
                amp, modulator_amp, pan;
```

```
		carrier_freq.spec_(\freq);
		modulator_freq.spec_(ControlSpec(0.1, 20000, \exp, 0, 1));
		amp.spec_(\amp);
		modulator_amp.spec_(\amp);
		pan.spec_(\pan);

		thisDisplay.synthDef_({ arg carrier_freq, modulator_freq,
					       amp, modulator_amp, pan;

			var  modulator, panner, result;

			modulator = SinOsc.ar(modulator_freq,
					mul: modulator_amp);
			result = SinOsc.ar(carrier_freq,
					mul: modulator * amp);
			panner = Pan2.ar(result, pan);
			Out.ar(0, panner);
		}, [\carrier_freq, carrier_freq, \modulator_freq,
			modulator_freq, \amp, amp, \modulator_amp, modulator_amp,
			\pan, pan]);

		thisDisplay.name_("Ring Modulation");
	}).show;
)
```

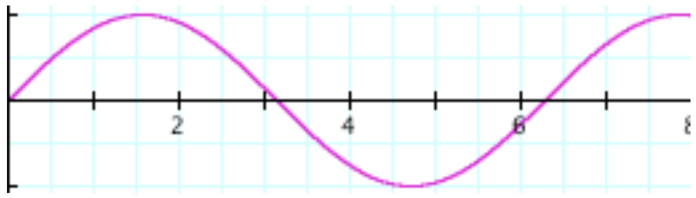
When you move both frequencies into the audio range, you should hear a complicated timbre. It is actually two tones. Ring Modulation produces sum and difference frequencies. If we set the carrier to 400 Hz and the modulator to 500 Hz, the resulting frequencies will be 100Hz and 900 Hz. (ROBERTS) “These products are called ‘sidebands'.” (ORTON/R) The result is easy and simple as long as we stick with sine waves, but when we switch to a more complicated waveform, one with overtones, the partials of the more complex wave all also create side bands. (ORTON/R) "[Each] partial of the one input will be added to and subtracted from each partial of the other." (ROBERTS) These side bands are "not related to the harmonic series." (ORTON/R) This creation of enharmonic partials is what gives ring modulation its noisy reputation.

In ring modulation, the modulating frequency goes between 1 and -1. This means that when both the modulator and the carrier are negative, the result is positive.

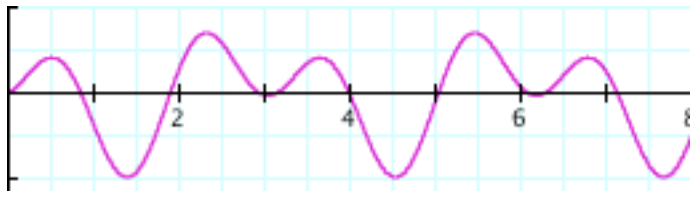
When these two frequencies are ring modulated:



and



The result is



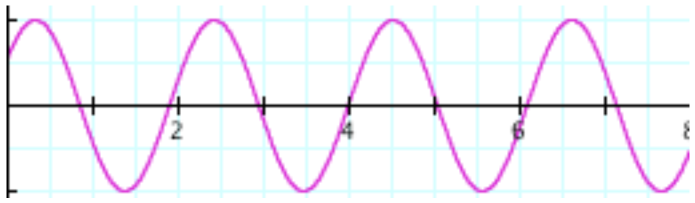
Notice that when both frequencies are negative the result is positive.

Amplitude Modulation

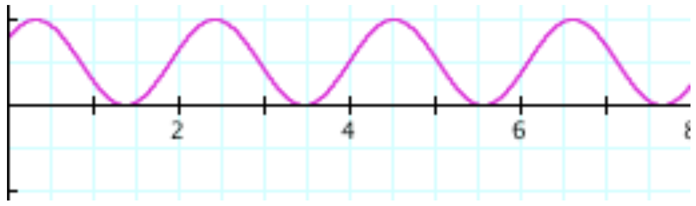
Ring Modulation is very closely linked to **Amplitude Modulation.** They are exactly the same except that in AM modulation, the modulator goes between 1 and 0 instead of between 1 and -1. There is never a circumstance where two negatives are multiplied together.

We divide the modulator signal by two to halve its amplitude. Then we add 0.5 to it so that it centers on 0.5 instead of on zero.

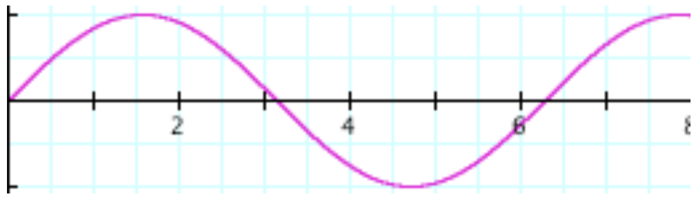
This transforms



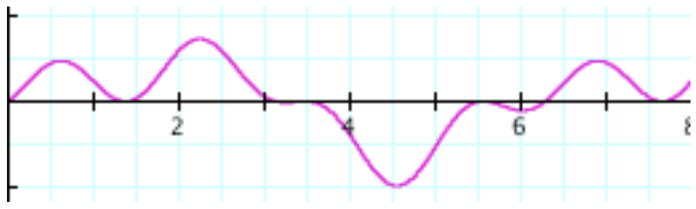
to



when you multiply that with



you get



Notice how that differs from the ring modulation graph in the previous section. Mathematically, the transformation is:

```
modulator = (modulator / 2) + 0.5;
```

Since modulator is a SinOsc, we can use the mul and add arguments.

```
modulator = SinOsc.ar(modulator_freq, 0, modulator_amp /2, 0.5);
```

Nicole's latest assignment is to create a display that does both ring modulation and amplitude modulation. She has noticed that dividing by 2 is the same as multiplying by 0.5. She realizes that the user can switch between RM and AM, if she uses a variable. She calls it offset.

```
modulator = SinOsc.ar(modulator_freq, 0, modulator_amp * offset,
                        offset);
```

When offset is 0.5, she gets AM modulation, but when the offset goes to zero, the amplitude goes to zero instead of one. She gets an idea.

```
modulator = SinOsc.ar(modulator_freq, 0, modulator_amp *
                        (1 - offset), offset);
```

When the offset is 0, the amplitude is 1 (times the modulator_amp). When the offset is 0.5, the amplitude is half. The offset doesn't have to be just 0 or 0.5. It can be any number in between. She creates a ControlSpec for the offset.

```
offset.spec_(ControlSpec(0, 0.5, \linear, 0, 0.5));
```

It is linear because you cannot have exponential ControlSpecs that have zero values.

Nicole's Display is below:

```
(

    Display.make({ arg thisDisplay, carrier_freq, modulator_freq,
                amp, modulator_amp, pan, offset;

        carrier_freq.spec_(\freq);
        modulator_freq.spec_(ControlSpec(0.1, 20000, \exp, 0, 1));
        amp.spec_(\amp);
        modulator_amp.spec_(\amp);
        pan.spec_(\pan);
        offset.spec_(ControlSpec(0, 0.5, \linear, 0, 0.5));

        thisDisplay.synthDef_({ arg carrier_freq, modulator_freq,
                        amp, modulator_amp, pan, offset;

            var  modulator, panner, result;

            modulator = SinOsc.ar(modulator_freq, 0, modulator_amp *
                        (1 - offset), offset);
            result = SinOsc.ar(carrier_freq,
                    mul: modulator * amp);
            panner = Pan2.ar(result, pan);
            Out.ar(0, panner);
        }, [\carrier_freq, carrier_freq, \modulator_freq,
            modulator_freq, \amp, amp, \modulator_amp, modulator_amp,
            \pan, pan, \offset, offset]);

        thisDisplay.name_("Amplitude / Ring Modulation");
    }).show;
)
```

When the offset is at 0.5, you are hearing amplitude modulation. When it is a 0, you are hearing ring modulation. By moving it between, you can hear sounds between the two. You should notice a third tone emerging as the offset approaches 0.5. Amplitude modulation produces the same two sidebands as ring modulation, but in AM, the carrier frequency is also audible. When the offset is at 0.5, the amplitude of the sidebands is half the amplitude of the carrier or less. (PENDER)

FM and Phase Modulation

FM is an idea used in radio and has been around for a good long while. Musical applications are somewhat newer. Nicole's analog synthesizers can do FM to create complicated wave shapes. Using digital FM to model physical instruments, however, dates from the 1970's. "Frequency Modulation (or FM) synthesis was discovered and introduced by John Chowning at Stanford around 1973." This technology was licensed to Yamaha and was the basis for the wildly popular Yamaha DX7 Synthesizer and the SoundBlaster card in the PCs of yore. ('FM synthesis on SND')

In FM synthesis, the modulator signal modulates the carrier frequency. That is, the pitch of the carrier is modified according to the wave form of the modulator. This is like vibrato, where a player moves their pitch up and down, modulating it over a regular period. We can use the properties of the modulator as the sole way to control the carrier.

```
modulator = SinOsc.ar(modulator_freq, mul:modulator_amp,
                      add: modulator_add);
carrier = SinOsc.ar(modulator, mul: carrier_amp);
```

If you think of FM as vibrato, `modulator_add` is the center frequency around which we are having vibrato. `modulator_amp` controls the depth of the vibrato and `modulator_freq` controls the speed of the vibrato.

```
(

    Display.make({ arg thisDisplay, modulator_freq, modulator_amp,
                 modulator_add, amp, pan;

        modulator_freq.sp(1, 0, 20000);
        modulator_amp.sp(1, 0, 20000);
        modulator_add.sp(0, 0, 20000);
        amp.spec_(\amp);
        pan.spec_(\pan);

        thisDisplay.synthDef_({ arg modulator_freq, amp,
                             modulator_amp, modulator_add, pan = 0;

            var modulator, carrier, panner;

            modulator = SinOsc.ar(modulator_freq, mul:modulator_amp,
                                 add: modulator_add);
            carrier = SinOsc.ar(modulator, mul: amp);

            panner = Pan2.ar(carrier, pan);
            Out.ar(0, panner);
        }, [\modulator_freq, modulator_freq, \amp, amp,
            \modulator_amp, modulator_amp, \modulator_add,
            modulator_add, \pan, pan]);

        thisDisplay.name_("Frequency Modulation");
    }).show;
)
```

Notice that, when the modulator_freq is at in the audio range, changing the modulator_amp changes the number of **side bands**. The effect of the modulator_amp, though, varies as you move the modulator_freq around.

Phase modulation is a type of FM. The SinOsc UGen takes a phase argument. We pass the modulator signal to the phase argument of the carrier.

```
    modulator = SinOsc.ar(modulator_freq, mul: modulator_amp);
    carrier = SinOsc.ar(carrier_freq, modulator, amp);
```

Notice that with phase modulation, the modulator_freq does not change the number of sidebands, only the frequency of them. The modulator_amp is what changes the number of side bands.

```
(

    Display.make({arg thisDisplay, carrier_freq, modulator_freq,
                amp, modulator_amp, pan;

        carrier_freq.spec_(\freq);
        modulator_freq.sp(1, 1, 20000, warp:'exponential');
        amp.spec_(\amp);
        modulator_amp.spec(\amp, 1);
        pan.spec_(\pan);


        thisDisplay.synthDef_({ arg carrier_freq, modulator_freq,
            amp, modulator_amp, pan;
            var modulator, carrier, panner;

            modulator = SinOsc.ar(modulator_freq,
                            mul: modulator_amp);
            carrier = SinOsc.ar(carrier_freq, modulator, amp);
            panner = Pan2.ar(carrier, pan);
            Out.ar(0, panner);
        }, [\carrier_freq, carrier_freq, \modulator_freq,
            modulator_freq, \amp, amp, \modulator_amp,
            modulator_amp, \pan, pan]);

        thisDisplay.name_("Phase Modulation");
    }).show;
)
```

Like AM, FM changes timbres by producing additional tones, known as side bands. Thanks to Chowning at Stanford, we know how these sidebands work.

```
Side band freq = carrier freq + (n * modulator freq)
```

n is any whole number. Let's imagine that we have a carrier of 400 Hz and a modulator of 100 Hz. The first whole number is 0. `400 + (0 * 100) = 400`. Then is 1. `400 + (1 * 100) = 500`. Then, also a whole number is -1. `400 + (-1 * 100) = 400- 100 = 300`. Then is 2 and -2. 400 + (2 * 100) = 600. 400 + (-2 * 100) = 200. Theoretically, n goes from negative infinity to positive infinity. As we've seen the amplitude of these sidebands in regular FM depends on the amplitude of the modulation signal AND the ratio between the carrier frequency and the modulator frequency. In phase modulation, the amplitude of sidebands depends only on the amplitude of the modulation signal.

“If the ratio between the frequencies [of the modulator] and carrier is simple (1:1, 2:1, 3:2 etc.) the sidebands generated will be in harmonic series, and the complex tones produced will resemble the overtone structures of real instruments.” (ROBERTS) Conversely, non-related frequencies will result in noisy, electronic-y sounds. You get a lot of frequencies from just two oscillators in FM, whereas, with RM, AM or additive synthesis, you would need many oscillators. FM and PM are effiecient ways to generate complicated tones.

Other Modulation

Pulse Width Modulation uses the kwidth argument of Pulse.ar

```
modulator = SinOsc.ar(modulator_freq, mul: modulator_amp);
carrier = Pulse.ar(carrier_freq, modulator, amp);
```

Any type of UGen or value can be used to modify the kwidth, as any UGen may be passed as (almost) any argument to any UGen. Many digital synthesizers have used pulse width modulation, often with square waves modifying the widths of other square waves.

Problems

1. Write a Display that AM and ring modulates two oscillators that are not sine waves.

2. Write a Display that uses a non-sine oscillator as a frequency modulator.

3. Write a Display that uses a non-sine oscillator as a phase modulator.

4. A chaos patch is one in which three oscillators FM modulate each other. Write a Phase modulation chaos patch, so that oscillator1 modifies the phase of oscillator2, 2 modifies 3 and 3 modifies 1.

5. Design your own modulating display using any number of oscillators in any configuration. Remember to avoid DC bias.

6. Use envelopes to control amplitudes of modulating frequencies. You can find example code for using envelopes in displays in chapter 5.

Project

Write a one or two minute piece using two different kinds of modulation.