March 17, 2024 08:46

Synthesis of simple music signals

1 Abstract

These notes are a quick introduction to synthesis of musical signals. They focus on digital wavetable synthesis, because one can apply this method easily to recordings actual instruments using up sampling, down sampling, and the circle of fifths. These notes also discuss additive synthesis, a method for creating artificial musical instruments that provides direct control over timbre. Subtractive synthesis and FM synthesis are mentioned briefly. This material can help improve synthesizer components of Project 3.
(Based on notes written originally by Prof. Andrew E. Yagle.)

2 Background

There are at least four major approaches to synthesis of musical signals. Two (FM and subtractive synthesis) apply primarily to analog signal processing, and are just summarized at the end. Additive and wavetable synthesis were developed originally for analog signal processing, but can also be used in digital signal processing. These methods will be our focus.

3 Wavetable synthesis

One way to synthesize music electronically is to record on tape short snippets of actual musical instruments playing a note. Then during playback, one can speed up or slow down the tape to vary the pitch of the note. Various instruments playing various notes can then be cut and pasted together (literally, using tape splicing!) or dubbed together to create music. The analog version uses a variable speed motor to drive the tape.

The same idea can be used in digital signal processing. Obviously it is easier to add and concatenate (string together) signals digitally than using tape. But how do we alter the pitch of a recorded instrument digitally? By applying multirate filtering (downsampling and upsampling), discussed next.

3.1 Multirate Filtering I: Downsampling–Increasing frequency by an integer factor

Doubling or tripling the pitch of a digital recording is simple, *provided all of the increased frequencies are less than half of the sampling rate.* Simply omit every other sample to double the pitch, and omit two out of three samples to triple the pitch. A moment's thought shows that this is equivalent to reducing the sampling rate by a factor of two or three. The effect of this modification is to double or triple the frequency of each harmonic in the Fourier series expansion of the signal, doubling or tripling the pitch of the musical signal. This "decimation" also reduces the duration of the signal, so the duration of the original signal must be increased, often by concatenation.

Basic `Julia` commands for downsampling a digital signal `x` (while maintaining the same duration) are:

- to double the pitch: `y = [x; x][1:2:end]`
- to triple the pitch: `y = [x; x; x][1:3:end]`

To see why downsampling works, consider its effect on an f-Hertz sinusoid sampled at S $\frac{\text{Sample}}{\text{Second}}$:

$$x(t) = A\cos(2\pi ft) \Longrightarrow x[n] = x(t)\Big|_{t=\frac{n}{S}} = A\cos\left(2\pi\frac{f}{S}n\right), \quad n = \ldots, -2, -1, 0, 1, 2, \ldots.$$

The downsampled signal (where we omit $x[n]$ for n odd) is then

$$y[n] = x[2n] = A\cos\left(2\pi\frac{f}{S}(2n)\right) = A\cos\left(2\pi\frac{2f}{S}n\right), \quad n = \ldots, -2, -1, 0, 1, 2, \ldots.$$

These equalities show immediately that downsampling doubles frequency.

An alternative approach to pitch doubling is to use FFT operations based on $f = k/T$:

```
# this works for sustained sounds, not for attack/release
function pitch_doubler(x)
    N = length(x)
    mod(N,4) == 0 || throw("N must be multiple of 4")
    Fall = fft(x) # entire spectrum
    keep = [(1:N÷4+1); (3N÷4+2):N]
    Flow = Fall[keep] # keep low half of the frequencies
    y = (1/2) * real(ifft(Flow))
    return [y; y] # maintain signal length
end
```

3.2 Multirate Filtering II: Upsampling–Reducing frequencies by an integer factor

To halve the pitch of a digital recording is slightly more complicated. Instead of discarding samples, we must include additional samples that interpolate the given samples. Simply averaging two samples to get an interpolated sample value between them gives an approximate solution, but the *exact* solution for a band-limited signal is as follows.

- Insert a zero between each pair of given samples, giving $\{\ldots, x[-2], 0, x[-1], 0, x[0], 0, x[1], 0, x[2], \ldots\}$;
- Apply a digital low-pass filter the resulting signal (that is twice as long as the original signal) as follows:
- (1) Compute its DFT; (2) Set the middle half of the DFT values to zero; (3) Compute the inverse DFT;
- The resulting signal has its pitch and amplitude halved, but its duration is twice as long.

The following code is an easier way. `Julia` commands for upsampling by 2 a digital signal `x` of even length `N=length(x)` :

```
function pitch_halver(x) # decrease pitch by one octave
    N = length(x)
    mod(N,2) == 0 || throw("N must be multiple of 2")
    F = fft(x) # original spectrum
    Fnew = [F[1:N÷2]; zeros(N+1); F[(N÷2+2):N]]
    return 2 * real(ifft(Fnew))[1:N]
end
```

To see why upsampling works, consider again its effect on an f-Hertz sinusoid sampled at S $\frac{\text{Sample}}{\text{Second}}$. The signal with zeros inserted at odd-numbered times is

$$z[n] = \begin{cases} x[n/2] & n \text{ even} \\ 0 & n \text{ odd} \end{cases} = A\cos\left(2\pi\frac{f}{S}\frac{n}{2}\right)\frac{1+\cos(n\pi)}{2}, \quad n = \ldots, -2, -1, 0, 1, 2, \ldots.$$

Using our old trigonometry friend the product-to-sum identity:

$$2\cos(a)\cos(b) = \cos(a+b) + \cos(a-b) \tag{1}$$

with $a = 2\pi\frac{f/2}{S}n$ and $b = n\pi = 2\pi\frac{S/2}{S}n$ gives

$$z[n] = \frac{A}{2}\cos\left(2\pi\frac{f/2}{S}n\right) + \frac{A}{4}\cos\left(2\pi\frac{f+S}{2S}n\right) + \frac{A}{4}\cos\left(2\pi\frac{f-S}{2S}n\right) = \frac{A}{2}\cos\left(2\pi\frac{f/2}{S}n\right) + \frac{A}{2}\cos\left(2\pi\frac{S-f}{2S}n\right).$$

Note that $\cos\left(2\pi\frac{f+S}{2F}n\right) = \cos\left(2\pi\frac{f-S}{2F}n\right) = \cos\left(2\pi\frac{S-f}{2F}n\right)$ because $\cos(x) = \cos(x-2\pi n) = \cos(-x)$. This shows that filtering out the sinusoid with frequency above $S/4$ leaves a sinusoid with halved frequency $f/2$. Repeating this argument for each harmonic in a Fourier series shows that upsampling halves frequency.

3.3 Circle of fifths

While we can apply upsampling and downsampling to alter pitch by a factor of any rational number, it is preferable to keep the factors small. Fortunately, we can alter any of the 12 semitones to any other semitone, by repeatedly downsampling by 2 and upsampling by 3, and by using the circle of fifths.

The circle of fifths is due to two numerical facts:

- $3^{12} = 531,441 \approx 524,288 = 2^{19} \implies \frac{3}{2} \approx 2^{7/12}$;
- 7 and 12 are relatively prime (no common factor).

As a result of these two facts, we have the following table, starting with note "A" at the *right* edge:

$\frac{\text{Frequency}}{440\text{ Hz}}$	$\frac{(\frac{3}{2})^0}{2^0}$	$\frac{(\frac{3}{2})^1}{2^0}$	$\frac{(\frac{3}{2})^2}{2^1}$	$\frac{(\frac{3}{2})^3}{2^1}$	$\frac{(\frac{3}{2})^4}{2^2}$	$\frac{(\frac{3}{2})^5}{2^2}$	$\frac{(\frac{3}{2})^6}{2^3}$	$\frac{(\frac{3}{2})^7}{2^4}$	$\frac{(\frac{3}{2})^8}{2^4}$	$\frac{(\frac{3}{2})^9}{2^5}$	$\frac{(\frac{3}{2})^{10}}{2^5}$	$\frac{(\frac{3}{2})^{11}}{2^6}$	$\frac{(\frac{3}{2})^{12}}{2^7}$
Hertz	440	660	495	742	557	835	626	470	705	529	793	595	446
Note	A	E	B	F#	C#	G#	D#	A#	F	C	G	D	A

This shows that by downsampling by 2 (sometimes twice) and upsampling by 3 repeatedly, we can obtain close approximations to all 12 semitones from any single semitone. So given a recording of an instrument playing any semitone, we can alter its pitch to any of the other semitones. Octave changes are handled by separate upsampling or downsampling by 2. When we *reduce* frequencies by upsampling, no aliasing occurs. When we *increase* frequencies by downsampling, aliasing can occur if the original frequency is too high.

We can downsample by 2 and upsample by 3 (to reduce frequencies by 2/3) in a single step as follows:

```
function down2_up3(x) # down-sample by 2, up-sample by 3.
    N = length(x)
    mod(N,2) == 0 || throw("N must be multiple of 2")
    F = fft(x) # original spectrum
    Fnew = [F[1:N÷2]; zeros(N÷2+1); F[(N÷2+2):N]]
    return 3/2 * real(ifft(Fnew))[1:N]
end
```

This approach is called the circle of fifths because arranging the 12 semitones in a circle, like hours on a clock face, and taking repeated jumps of 7 "hours" each, we can reach all 12 "hours." A fifth is the interval between the first note and the fifth note in a major or minor scale (such as A to E above), and is equal to 7 semitones. The circle

of fifths is actually a relic of the era when pitches were related exactly by various ratios of small integers, instead of the constant ratios of $2^{1/12}$ used today.

4 Additive synthesis

4.1 Specification of Fourier Series amplitudes

Once we accept that musical notes can be represented by Fourier series of a fundamental and higher harmonics, also called overtones or partials (partials can vary over time), a way to synthesize music is to select Fourier Series amplitudes $\{c_k\}$ and then generate signals using the synthesis formula:

$$x(t) = c_1 \cos(2\pi f t) + c_2 \cos(2\pi 2 f t) + \cdots \Longrightarrow x[n] = c_1 \cos\left(2\pi \frac{f}{S} n\right) + c_2 \cos\left(2\pi \frac{2f}{S} n\right) + c_3 \cos\left(2\pi \frac{3f}{S} n\right) + \cdots . \tag{2}$$

- This generates a pitch at frequency f Hertz for a sampling rate $S \frac{\text{Sample}}{\text{Second}}$;
- The amplitudes c_k determine the timbre of the sound (varies by instrument);
- Phases are omitted, because the human ear does not perceive them for monophonic music;
- The number of terms, $\left\lfloor \frac{S}{2f} \right\rfloor$ is finite because kf may not exceed $S/2$;
- Changing the frequency is merely a matter of using a different value of f.
- Hammond organs and pipe organs do this *physically*: opening stops varies the c_k values.

To play "A" (fundamental frequency: 440 Hertz) sampled at $S \frac{\text{Sample}}{\text{Second}}$ using additive synthesis:

```
x = cos.(2π*440*(1:N)*(1:length(c))'/S) * c
```

This command is deceptively simple. It implements the summation shown in (2) even though no `sum` or `+` appears explicitly. It works because `c` is a vector, and it exploits the array operations built into `Julia`. The part that is `2*pi*440*(1:N)*(1:length(c))'/S` generates a 2D array that has the values of $2\pi 440 \frac{k}{S} n$ for `N` values of n and for `length(c)` values of c_k. Then `cos.( )` computes the cosine of each element of that array one-by-one. Finally, multiplying by the vector `* c` adds up the `length(c)` sinusoids to form one additively synthesized signal that one can play using `soundsc`.

Try listening to this for the following two vectors of Fourier series amplitudes c_k for `S = 44100; N = S/2`.

Factor	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	listen
1/60	24	9	6	10	1.8	4	2.5	.9	.9	.55	0	0	0	0	0	play
1/410	34	27	60	67	30	42	6	5.1	4.5	3.5	5	16	48	42	12.5	play

- `c = 1/60 * vec([ 24 9 6 10 1.8 4 2.5 .9 .9 .55 ])`
- `c = 1/410 * vec([ 34 27 60 67 30 42 6 5.1 4.5 3.5 5 16 48 42 12.5 ])`

The one line of code shown above will produce a sound with a certain timbre, but it will *not* sound like a realistic instrument because real instruments also have an envelope that describes the attack, decay, sustain, and release of the sound. A lecture will show how to incorporate envelopes into the synthesis process.

4.2 Generating sinusoids recursively

When using `Julia`, the Fourier series can easily be generated directly. However, if a DSP chip is used, computing all of the cosines is more expensive. Fortunately, we can use the trigonometric identity (1) yet again to eliminate repeated computation of cosines.

We want to generate $x[n] = A\cos\left(2\pi\frac{f}{S}n\right)$, a cosine of frequency f Hertz sampled at $S\frac{\text{Sample}}{\text{Second}}$. Setting $a = 2\pi\frac{f}{S}n$ and $b = 2\pi\frac{f}{S}$ in the trig identity and multiplying by A gives

$$x[n+1] + x[n-1] = 2\cos\left(2\pi\frac{f}{S}\right)x[n] \Longrightarrow x[n+1] = 2\cos\left(2\pi\frac{f}{S}\right)x[n] - x[n-1],$$

which we recognize as a rearrangement of the frequency estimation algorithm from Lab 2.

But we can also use this equation to compute $x[n+1]$ recursively from $x[n]$ and $x[n-1]$. Starting with $x[0] = A$ and $x[1] = A\cos\left(2\pi\frac{f}{F}\right)$, we can plug into this equation and compute $x[2], x[3], \ldots$. Each step requires only a single multiplication and subtraction; any chip can perform this action very quickly. This approach avoids explicit computation of cosines, except for the single constant $2\cos\left(2\pi\frac{f}{S}\right)$ needed to initialize the recursion. (This method also works with nonzero phase values θ.)

5 Subtractive and FM synthesis

Historically these were primarily analog techniques, but we include them for completeness.

5.1 Subtractive synthesis

The idea here is to use analog circuitry to generate a simple periodic signal, such as a square wave or sawtooth wave, and then use an analog filter to remove most of its harmonics. Moog synthesizers do this.

5.2 FM synthesis

The idea in FM (Frequency Modulation) synthesis is to use an FM signal that looks like

$$x(t) = A\sin(2\pi ft + I\sin(2\pi gt)),$$

where I is the modulation index and f and g are both frequencies. This is a simple way to generate complicated periodic signals that are rich in harmonics [**?**]. Another way is to use a nonlinear function such as $y(t) = \frac{|x(t)|}{1+|x(t)|}$.

5.3 Plucked string synthesis

One way to emulate the sound of a plucked string, is the Karplus-Strong method.

References

[1] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.