Time Varying Delay Effects

Julius Smith and Nelson Lee

RealSimple Project*
Center for Computer Research in Music and Acoustics (CCRMA)
Department of Music, Stanford University
Stanford, California 94305

June 5, 2008

It is often necessary for a delay line to vary in length. (Consider simulating a sound ray when either the source or listener is moving.) In this case, separate read and write pointers are normally used.

*Work supported by the Wallenberg Global Learning Network

Variable Delay Lines

Let A denote an array of length N. Then we can implement an M-sample variable delay line in the C programming language as shown in Fig. ??. We require, of course, $M \leq N$.

The M-sample variable delay line using separate read- and write-pointers:

```
static double A[N];
static double *rptr = A; // read ptr
static double *wptr = A; // write ptr

double setdelay(int M) {
    rptr = wptr - M;
    while (rptr < A) { rptr += N }
}

double delayline(double x)
{
  double y;
  A[wptr++] = x;
  y = A[rptr++];
  if ((wptr-A) >= N) { wptr -= N }
  if ((rptr-A) >= N) { rptr -= N }
  return y;
}
```

The Synthesis Tool Kit, Version 4 [?] contains the C++ class "Delay" which implements this type of variable (but

non-interpolating) delay line. There are additional subclasses which provide *interpolating reads* by various methods. In particular, the class DelayL implements continuously variable delay lengths using *linear interpolation*. The code listing in Fig. ?? can be modified to use linear interpolation by replacing the line

```
y = A[rptr++];
```

with

```
long rpi = (long)floor(rptr);
double a = rptr - (double)rpi;
y = a * A[rpi] + (1-a) * A[rpi+1];
rptr += 1;
```

To implement a *continuously* varying delay, we add a "delay growth parameter" g to the delayline function in Fig. ??, and change the line

```
rptr += 1; // pointer update
```

above to

```
rptr += 1 - g; // pointer update
```

When g is 0, we have a fixed delay line. When $g > 0$, the delay grows g samples per sample, which we may also interpret as seconds per second, *i.e.*, $\dot{D}_t = g$. In §?? on page ??, this will be applied to simulation of the *Doppler effect*.

Delay-Line Interpolation

When delay lines need to vary smoothly over time, some form of *interpolation* between samples is usually required to avoid "zipper noise" in the output signal as the delay length changes. There is a hefty literature on "fractional delay" in discrete-time systems, and the survey in [?] is highly recommended.

This section will describe the most commonly used cases. *Linear interpolation* is perhaps most commonly used because it is very straightforward and inexpensive, and because it sounds very good when the signal bandwidth is small compared with half the sampling rate. For a delay line in a nearly lossless feedback loop, such as in a vibrating string simulation, *allpass interpolation* is usually a better choice since it costs the same as linear interpolation in the first-order case and has no gain distortion. (Feedback loops can be very sensitive to gain distortions.) Finally, in Appendix ??, some higher order interpolation schemes are outlined.

Linear Interpolation

Linear interpolation works by effectively drawing a straight line between two neighboring samples and returning the appropriate point along that line.

More specifically, let η be a number between 0 and 1 which represents how far we want to interpolate a signal y between time n and time $n+1$. Then we can define the linearly interpolated value

$\hat{y}(n+\eta)$ as follows:

$$\hat{y}(n+\eta) = (1-\eta)\cdot y(n) + \eta \cdot y(n+1)$$

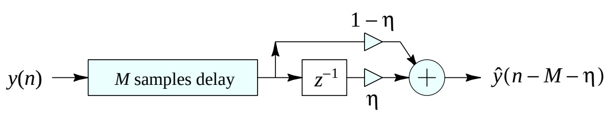
For $\eta = 0$, we get exactly $\hat{y}(n) = y(n)$, and for $\eta = 1$, we get exactly $\hat{y}(n+1) = y(n+1)$. In between, the interpolation error $|\hat{y}(n+\eta) - y(n+\eta)|$ is nonzero, except when $y(t)$ is a linear function between $y(n)$ and $y(n+1)$.

Note that by factoring out η, we can obtain a *one-multiply* form,

$$\hat{y}(n+\eta) = y(n) + \eta \cdot [y(n+1) - y(n)]\,.$$

Thus, the computational complexity of linear interpolation is one multiply and two additions per sample of output.

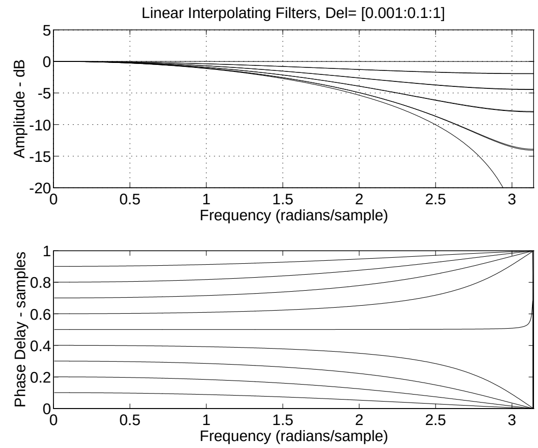
A linearly interpolated delay line is depicted in Fig. ??.



Linearly interpolated delay line.

The C++ class implementing a linearly interpolated delay line in the Synthesis Tool Kit (STK) is called `DelayL`.

The frequency response of linear interpolation for fixed fractional delay (η fixed in Fig. ??) is shown in Fig. ??. From inspection of Fig. ??, we see that linear interpolation is a one-zero FIR filter. When used to provide a fixed fractional delay, the filter is linear and time-invariant (LTI). When the delay provided changes over time, it is a linear time-varying filter.



Linear interpolation frequency responses for delays between 0 and 1. Note the higher accuracy at low frequencies, reaching zero error at dc for all fractional delays.

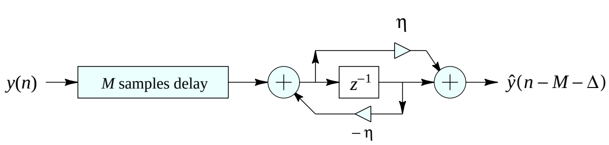
Linear interpolation sounds best when the signal is *oversampled*. Since natural audio spectra tend to be relatively concentrated at low frequencies, linear interpolation tends to sound very good at high sampling rates.

When interpolation occurs inside a *feedback loop*, such as in digital waveguide models for vibrating strings, errors in the amplitude response can be highly audible (particularly when the loop gain is close to 1, as it is for steel strings, for example). In these cases, it is

possible to eliminate amplitude error (at some cost in delay error) by using an *allpass filter* for delay-line interpolation.

First-Order Allpass Interpolation

A delay line interpolated by a first-order allpass filter is drawn in Fig. ??.



Allpass-interpolated delay line.

Intuitively, ramping the coefficients of the allpass gradually "grows" or "hides" one sample of delay. This tells us how to handle resets when crossing sample boundaries.

The difference equation is

$$\begin{aligned}\hat{x}(n-\Delta) \stackrel{\Delta}{=} y(n) &= \eta \cdot x(n) + x(n-1) - \eta \cdot y(n-1)\\ &= \eta \cdot [x(n) - y(n-1)] + x(n-1).\end{aligned}$$

Thus, like linear interpolation, first-order allpass interpolation requires only one multiply and two adds per sample of output.

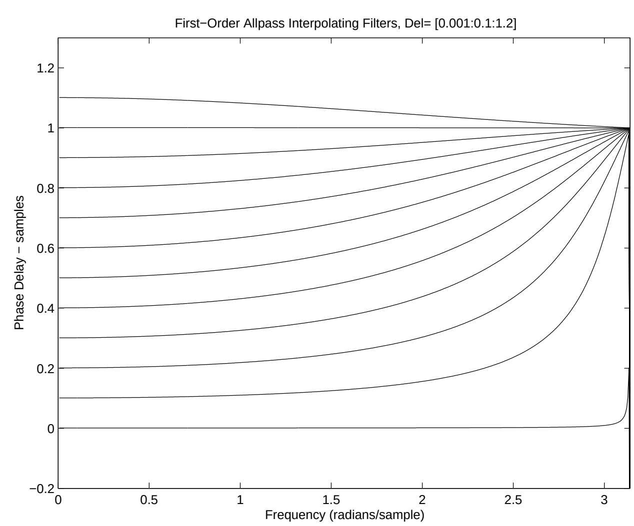
The frequency response is

$$H(z) = \frac{\eta + z^{-1}}{1 + \eta z^{-1}}. \tag{1}$$

At low frequencies ($z \to 1$), the delay becomes

$$\Delta \approx \frac{1-\eta}{1+\eta} \tag{2}$$

Figure ?? shows the *phase delay* of the first-order digital allpass filter for a variety of desired delays at dc. Since the amplitude response of any allpass is 1 at all frequencies, there is no need to plot it.



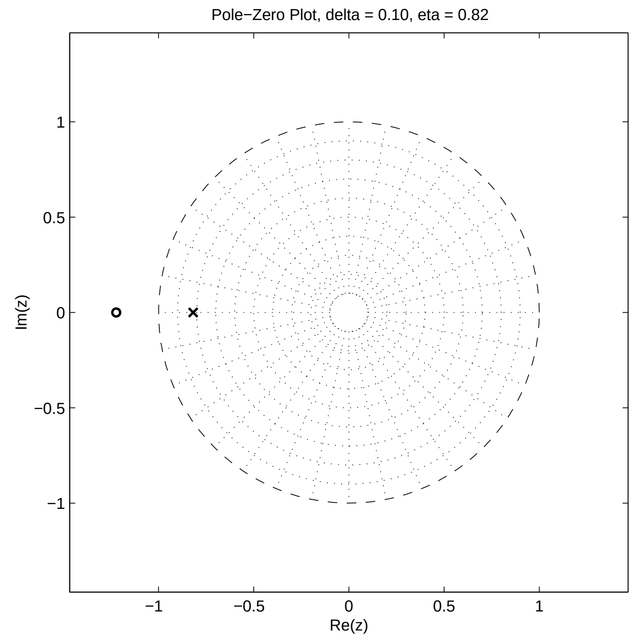
Allpass-interpolation phase delay for a variety of desired delays (exact at dc).

The first-order allpass interpolator is generally controlled by setting its dc delay to the desired delay. Thus, for a given desired delay Δ, the allpass coefficient is (from Eq. (2))

$$\eta \approx \frac{1-\Delta}{1+\Delta}$$

From Eq. (1), we see that the allpass filter's pole is at $z=-\eta$, and its zero is at $z=-1/\eta$. A pole-zero diagram for $\Delta=0.1$ is given in Fig. ??. Thus, zero delay is provided by means of a *pole-zero cancellation*! Due to inevitable round-off errors, pole-zero cancellations are to be avoided in practice. For this reason and others (discussed below), allpass interpolation is best used to provide a delay range lying wholly above zero, *e.g.*,

$$\Delta \in [0.1, 1.1] \;\longleftrightarrow\; \eta \in [-0.05, 0.82]$$



Allpass-interpolator pole-zero diagram for $\Delta=0.1$.

Note that, unlike linear interpolation, allpass interpolation is not suitable for "random access" interpolation in which interpolated values may be requested at any arbitrary time in isolation. This is because the allpass is *recursive* so that it must run for enough

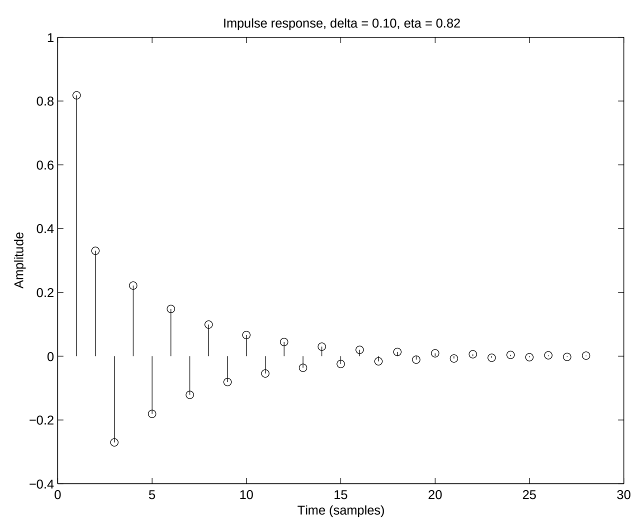
samples to reach steady state. However, when the impulse response is reasonably short, as it is for delays near one sample, it can in fact be used in "random access mode" by giving it enough samples with which to work.

The STK class implementing allpass-interpolated delay is `DelayA`.

Minimizing the Transient Response of First-Order Allpass Interpolation

In addition to approaching a pole-zero cancellation at $z=-1$, another undesirable artifact appears as $\Delta \to 0$: The *transient response* also becomes long when the pole at $z=-\eta$ gets close to the unit circle.

A plot of the impulse response for $\Delta=0.1$ is shown in Fig. ??. We see a lot of "ringing" near half the sampling rate. We actually should expect this from the *nonlinear phase distortion* which is clearly evident near half the sampling rate in Fig. ??. We can interpret this phenomenon as the signal components near half the sampling rate being delayed by different amounts than other frequencies, therefore "sliding out of alignment" with them.



Impulse response of the first-order allpass interpolator for $\Delta=0.1$.

For audio applications, we would like to keep the impulse-response duration short enough to sound "instantaneous." That is, we do not wish to have audible "ringing" in the time domain near $f_s/2$. For high quality sampling rates, such as larger than $f_s=40$ kHz, there is no issue of direct audibility, since the ringing is above the range of human hearing. However, it is often convenient, especially for research prototyping, to work at lower sampling rates where $f_s/2$ is audible. Also, many commercial products use such sampling rates to save costs.

Since the time constant of decay, in samples, of the impulse response of a pole of radius R is approximately

$$\frac{\tau}{T} \approx \frac{1}{1-R},$$

and since a 60-dB decay occurs in about 7 time constants ("t_{60}") [?, p. 38], we can limit the pole of the allpass filter to achieve any prescribed specification on maximum impulse-response duration.

For example, suppose 100 ms is chosen as the maximum t_{60} allowed at a sampling rate of $f_s = 10,000$. Then applying the above constraints yields $\eta \leq 0.993$, corresponding to the allowed delay range $[0.00351, 1.00351]$.

1 Large Delay Changes

When implementing large delay length changes (by many samples), a useful implementation is to *cross-fade* from the initial delay line configuration to the new configuration:

- Computational requirements are doubled during the cross-fade.
- The cross-fade should occur over a time interval long enough to yield a smooth result.
- The new delay interpolation filter, if any, may be initialized in advance of the cross-fade, for maximum smoothness. Thus, if the transient response of the interpolation filter is N samples, the new delay-line + interpolation filter can be "warmed up" (executed) for N time steps before beginning the cross-fade. If

the cross-fade time is long compared with the interpolation filter duration, "pre-warming" is not necessary.
- This is not a true "morph" from one delay length to another since we do not pass through the intermediate delay lengths. However, it avoids a potentially undesirable Doppler effect.
- A single delay line can be *shared* such that the cross-fade occurs from one *read-pointer* (plus associated filtering) to another.

Specific Time-Varying Delay Effects

Time varying delay lines are fundamental building blocks for delay effects, synthesis algorithms, and computational acoustic models of musical instruments.

In the category of *delay effects*, variable delay lines are used for

- Phasing
- Flanging
- Chorus
- Leslie
- Reverb

(While reverberators need not be time varying, nowadays they typically are [?, ?].)

In *digital waveguide synthesis*, variable delay lines are used for

- Vibrating strings (guitars, violins, ...)
- Woodwind bores
- Horns
- Tonal percussion (rods, membranes)

The following sections will elaborate on the use of variable delay lines for *effects*. Their use in digital waveguide models will be deferred to a later section.

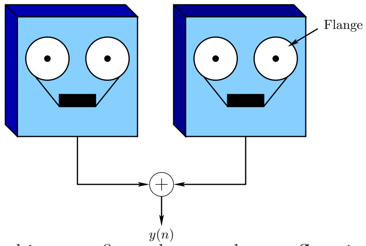
Flanging

Flanging is a delay effect that has been available in recording studios since at least the 1960s. Surprisingly little literature exists, although there is some [?, ?, ?, ?, ?, ?].

According to lore [?, ?], the term "flanging" arose from the way the effect was originally achieved by two tape machines set up to play the same tape in unison, with their outputs are added together (mixed equally), as shown in Fig. 1. To achieve the flanging effect, the *flange* of one of the supply reels is touched lightly to make it play a littler slower. This causes a *delay* to develop between two tape machines. The flange is released, and the flange of the other supply reel is touched lightly to slow it down. This causes the delay to gradually disappear and then begin to grow again in the opposite direction. The delay is kept below the threshold of echo perception (*e.g.*, only a few milliseconds in each direction). The process is repeated as desired, pressing the flange of each supply reel in alternation. The flanging effect has been described as a kind of "whoosh" passing subtly through the sound.[1] The effect is also compared to the sound of a jet passing overhead, in which the direct signal and ground reflection arrive at a varying relative delay [?]. If flanging is done rapidly enough, an audible Doppler shift is introduced which approximates the "Leslie" effect commonly used for organs (see §??).

Flanging is modeled quite accurately as a feedforward comb filter, as discussed in §??, in which the delay M is varied over time. Figure 1 depicts such a model. The input-output relation for a basic flanger

[1] For sound examples, see http://www.harmony-central.com/Effects/Articles/Flanging/.

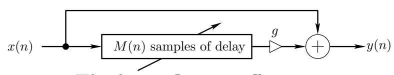


Two tape machines configured to produce a *flanging effect*.

can be written as

$$y(n) = x(n) + gx[n - M(n)] \qquad (3)$$

where $x(n)$ is the input signal amplitude at time $n = 0, 1, 2, \ldots$, $y(n)$ is the output at time n, g is the "depth" of the flanging effect, and $M(n)$ is the length of the delay-line at time n. The delay length $M(n)$ is typically varied according to a triangular or sinusoidal waveform. We may say that the delay length is *modulated* by an "*LFO*" (Low-Frequency Oscillator). Since $M(n)$ must vary smoothly over time, it is clearly necessary to use an *interpolated delay line* to provide non-integer values of M in a smooth fashion.



The basic flanger effect.

As shown in Fig. ?? on page ??, the frequency response of Eq. (3) has a "comb" shaped structure. For $g > 0$, there are M *peaks* in the

frequency response, centered about frequencies

$$\omega_k^{(p)} = k\frac{2\pi}{M}, \quad k = 0, 1, 2, \ldots, M - 1.$$

For $g = 1$, the peaks are maximally pronounced, with M *notches*[2] occurring between them at frequencies $\omega_k^{(n)} = \omega_k^{(p)} + \pi/M$. As the delay length M is varied over time, these "comb teeth" squeeze in and out like the pleats of an accordion. As a result, the spectrum of any sound passing through the flanger is "massaged" by a variable comb filter.

As is evident from Fig. ?? on page ??, at any given time there are $M(n)$ *notches* in the flanger's amplitude response (counting positive- and negative-frequency notches separately). The notches are thus spaced at intervals of f_s/M Hz, where f_s denotes the sampling rate. In particular, the *notch spacing is inversely proportional to delay-line length*.

The time variation of the delay-line length $M(n)$ results in a "sweeping" of uniformly-spaced notches in the spectrum. The flanging effect is thus created by *moving notches* in the spectrum. Notch motion is essential for the flanging effect. Static notches provide some coloration to the sound, but an isolated notch may be inaudible [?]. Since the steady-state sound field inside an undamped *acoustic tube* has a similar set of uniformly spaced notches (except at the ends), a static row of notches tends to sound like being inside an acoustic tube.

[2] The term *notch* here refers to the elimination of sound energy at a single frequency or over a narrow frequency interval. Another term for this is "*null*".

Flanger Speed and Excursion

As mentioned above, the delay-line length $M(n)$ in a digital flanger is typically modulated by a *low-frequency oscillator (LFO)*. The oscillator waveform is usually triangular, sinusoidal, or exponential (triangular on a log-frequency scale). In the sinusoidal case, we have the following delay variation:

$$M(n) = M_0 \cdot [1 + A\sin(2\pi f nT)]$$

where f is the "*speed*" (or "*rate*") of the flanger in cycles per second, A is the "excursion" or "sweep" (maximum delay swing) which is often not brought out as a user-controllable parameter, and M_0 is the average delay length controlling the average notch density (also not normally brought out as a user-controllable parameter).

Flanger Depth Control

To obtain a maximum effect, the depth control, g in Fig. 1, should be set to 1. A depth of $g = 0$ gives no effect.

Flanger Inverted Mode

A different type of maximum depth is obtained for $g = -1$. In this case, the peaks and notches of the $g = 1$ case trade places. In practice, the depth control g is usually constrained to the interval $[0, 1]$, and a sign inversion for g is controlled separately using a "phase inversion" switch.

In inverted mode, unless the delay M is very large, the bass response will be weak, since the first notch is at dc. This case usually sounds high-pass filtered relative to the "in-phase" case ($g > 0$).

As the notch spacing grows very large (M shrinks), the amplitude response approaches that of a first-order difference $y(n) = x(n) - x(n-1)$, which approximates a differentiator $y(t) = \frac{d}{dt}x(t)$. An ideal differentiator eliminates dc and provides a progressive high-frequency boost rising 6 dB per octave (specifically, the amplitude response is $|H(\omega)| = |\omega|$).

Flanger Feedback Control

Many modern commercial flangers have a control knob labeled "feedback" or "regen." This control sets the level of feedback from the output to the input of the delay line, thereby creating a *feedback comb filter* in addition to the feedforward comb filter, in the same manner as in the creation of a Schroeder allpass filter.

More generally, outputs of *any subset* of the allpass sections can be fed back to the input (in small amounts) to produce different sounding effects [?].

Summary of Flanging

In view of the above, we may define a *flanger* in general as any filter which modulates the frequencies of a set of *uniformly* spaced notches and/or peaks in the frequency response. The main parameters are

- Depth $g \in [0, 1]$ — controlling notch depth

- Speed f — speed of notch movement
- Phase — switch to subtract instead of adding the direct signal with the delayed signal

Possible additional parameters include

- Average Delay M_0
- Excursion or Sweep A — amount by which the delay-line grows or shrinks
- Feedback or Regeneration $a_M \in (-1, 1)$ — feedback coefficient from output to input

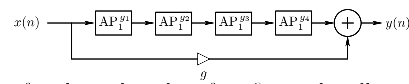
Note that flanging provides *only* uniformly spaced notches. This can be considered non-ideal for several reasons. First, the ear processes sound over a frequency scale that is more nearly logarithmic than linear [?]. Therefore, exponentially spaced notches (uniformly spaced on a log frequency scale) should sound more uniform perceptually. Secondly, the uniform peaks and notches of the flanger can impose a discernible "resonant pitch" on the program material, giving the impression of being inside a resonant tube. Third, it is possible for a periodic tone to be completely annihilated by harmonically spaced notches if the harmonics of the tone are unlucky enough to land exactly on a subset of the harmonic notches. In practice, exact alignment is unlikely; however, the signal loudness can be modulated to a possibly undesirable degree as the notches move through alignment with the signal spectrum. For this reason, flangers are best used with noise-like or inharmonic sounds. For harmonic signals, it makes sense to consider methods for creating *non-uniform* moving notches.

Phasing

The *phaser*, or *phase shifter*, is closely related to the flanger in that it also works by sweeping notches through the spectrum of the input signal. While the term phasing is sometimes used synonymously with flanging [?], commercial phase shifters have been observed to implement *nonuniformly spaced* notches.[3] We will therefore define a *phaser* as any linear filter which modulates the frequencies of a set of *non-uniformly spaced notches*, while a *flanger* will remain any device which modulates *uniformly spaced notches*.

Phasing with First-Order Allpass Filters

The block diagram of a typical inexpensive *phase shifter* for guitar players is shown in Fig. 1.[4] It consists of a series chain of first-order allpass filters,[5] each having a single time-varying parameter $g_i(n)$ controlling the pole and zero location over time, plus a feedforward path through gain g which is a fixed *depth control*.



Structure of a phaser based on four first-order allpass filters.

[3] The author discovered this by looking at the circuit for the MXR phase shifter in 1975.

[4] This is the basic architecture of the MXR phase shifter as well as the *Univibe* used by Jimi Hendrix [?], and described in detail at (`http://www.geofex.com/Article_Folders/univibe/uvfrindx.htm`).

[5] Moog has built a 12-stage phaser of this type [?] and up to 20 stages (10 notches) have been noted [?].

In analog hardware, the first-order allpass transfer function [?, Appendix C, Section 8][6] is

$$\mathsf{AP}_1^{\omega_b} \triangleq \frac{s - \omega_b}{s + \omega_b}. \tag{4}$$

In discrete time, the general first-order allpass has the transfer function

$$\mathsf{AP}_1^{g_i} \triangleq \frac{g_i + z^{-1}}{1 + g_i z^{-1}}.$$

We now consider the analog and digital cases, respectively.

Classic Analog Phase Shifters

Setting $s = j\omega$ in Eq. (4) gives the frequency response of the analog-phaser transfer function to be

$$H_a(j\omega) = \frac{j\omega - \omega_b}{j\omega + \omega_b}.$$

The phase response is readily found to be

$$\Theta_i(\omega) = \pi - 2\tan^{-1}\left(\frac{\omega}{\omega_b}\right).$$

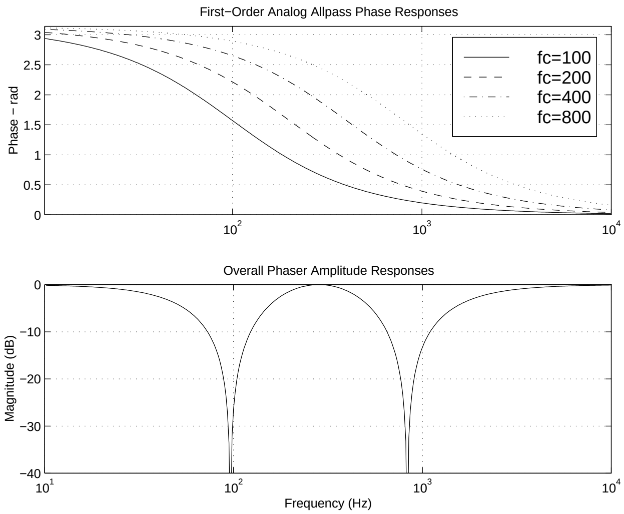
Note that the phase is always π at dc ($\omega = 0$), meaning each allpass section inverts at dc. Also, at $\omega = \infty$ (remember we're talking about analog here), we get a phase of zero. In between, the phase falls from π to 0 as frequency goes from 0 to ∞. In particular, at $\omega = \omega_b$, the phase has fallen exactly half way, to $\pi/2$. We will call $\omega = \omega_b$ the *break frequency* of the allpass section.[7]

[6] Available online at
`http://ccrma.stanford.edu/~jos/filters/Analog_Allpass_Filters.html`

[7] Saying that the frequency ω_0 is the *break frequency* for the one-pole term $H(s) = b/(s + \omega_0)$ is terminology from *classical control theory*. Below the break frequency, $H(s) \approx b/\omega_0$, and above, $H(s) \approx b/s$. On a log-log plot,

Figure ??a shows the phase responses of four first-order analog allpass filters with ω_b set to $2\pi[100, 200, 400, 800]$. Figure ??b shows the resulting normalized amplitude response for the phaser, for $g = 1$ (unity feedfoward gain). The amplitude response has also been normalized by dividing by 2 so that the maximum gain is 1. Since there is an even number (four) of allpass sections, the gain at dc is $1 + (-1)(-1)(-1)(-1) = 1$. Put another way, the initial phase of each allpass section at dc is π, so that the total allpass-chain phase at dc is 4π. As frequency increases, the phase of the allpass chain decreases. When it comes down to 3π, the net effect is a sign inversion by the allpass chain, and the phaser has a notch. There will be another notch when the phase falls down to π. Thus, four allpass sections give two notches. For each notch in the desired response we must add two new allpass sections.

the amplitude response $|H(j\omega)|$ may be approximated by a slope-zero line at height $G_0 = 20\log_{10}(|b|)$ from dc to ω_0, followed by an intersecting line with negative slope of 20 dB per decade for all higher frequencies. At the break frequency, the true gain is down 3dB from G_0, but far away from the break frequency, the piecewise-linear approximation is extremely accurate. Such an approximate amplitude response is called a *Bode plot*. Bode plots are covered in any introductory course on control systems design (also called the design of "servomechanisms").



(a) Phase reponses of first-order analog allpass sections with break frequencies at 100, 200, 400, and 800 Hz. (b) Corresponding phaser amplitude response.

From Fig. ??b, we observe that the first notch is near $f = 100$ Hz. This happens to be the frequency at which the first allpass pole "breaks," *i.e.*, $\omega = g_1$. Since the phase of a first-order allpass section at its break frequency is $\pi/2$, the sum of the other three sections must be approximately $2\pi + \pi/2$. Equivalently, since the first section has "given up" $\pi/2$ radians of phase at $\omega = g_1 = 2\pi 100$, the other three allpass sections *combined* have given up $\pi/2$ radians as well

(with the second section having given up more than the other two).

In practical operation, the break frequencies must *change dynamically*, usually periodically at some rate.

Classic Virtual Analog Phase Shifters

To create a *virtual analog* phaser, following closely the design of typical analog phasers, we must translate each first-order allpass to the digital domain. Working with the transfer function, we must map from s plane to the z plane. There are several ways to accomplish this goal [?]. However, in this case, an excellent choice is the *bilinear transform* (see §?? on page ??), defined by

$$s \leftarrow c\frac{z-1}{z+1} \tag{5}$$

where c is chosen to map one particular frequency to exactly where it belongs. In this case, c can be chosen for each section to map the *break frequency* of the section to exactly where it belongs on the digital frequency axis. The relation between the analog and digital frequency axes follows immediately from Eq. (5) as

$$\begin{aligned} j\omega_a &= c\frac{e^{j\omega_d T}-1}{e^{j\omega_d T}+1} \\ &= c\frac{e^{j\omega_d T/2}\left(e^{j\omega_d T/2}-e^{-j\omega_d T/2}\right)}{e^{j\omega_d T/2}\left(e^{j\omega_d T/2}+e^{-j\omega_d T/2}\right)} \\ &= jc\frac{\sin(\omega_d T/2)}{\cos(\omega_d T/2)} \\ &= jc\tan(\omega_d T/2). \end{aligned}$$

Thus, given a particular desired break frequency $\omega_a = \omega_d = \omega_b$, we can set

$$c = \omega_b \cot(\omega_b T/2).$$

Recall from Eq. (4) on page 23 that the transfer function of the first-order *analog* allpass filter is given by

$$H_a(s) = \frac{s-\omega_b}{s+\omega_b}$$

where ω_b is the break frequency. Applying the general bilinear transformation Eq. (5) gives

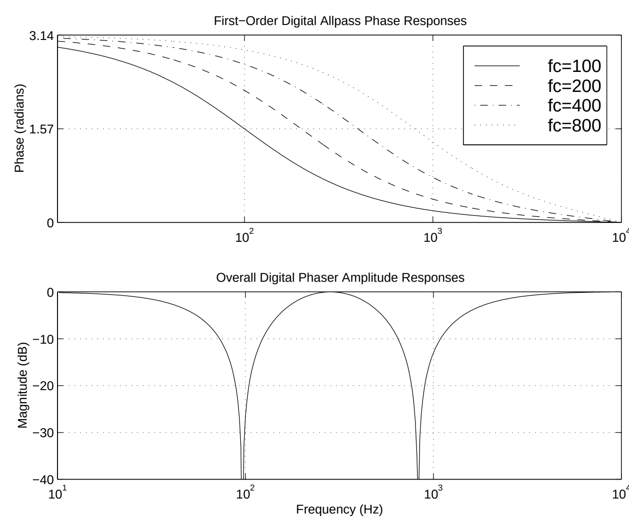
$$\begin{aligned} H_d(z) &= H_a\left(c\frac{1-z^{-1}}{1+z^{-1}}\right) = \frac{c\left(\frac{1-z^{-1}}{1+z^{-1}}\right)-\omega_b}{c\left(\frac{1-z^{-1}}{1+z^{-1}}\right)+\omega_b} \\ &\triangleq \frac{p_d - z^{-1}}{1-p_d z^{-1}} \end{aligned}$$

where we have denoted the pole of the digital allpass by

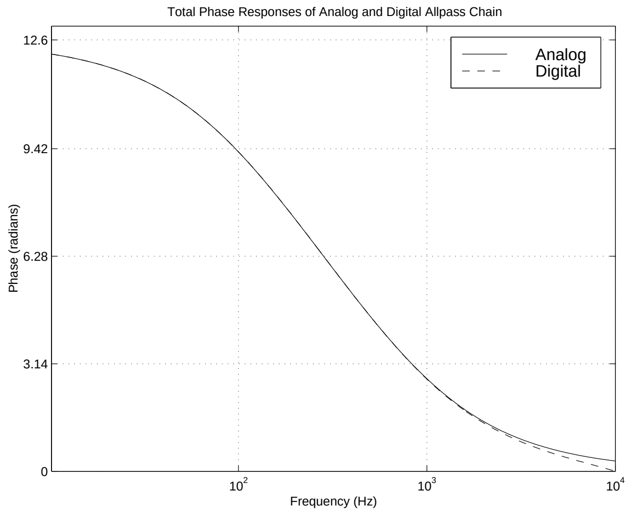
$$p_d \triangleq \frac{c-\omega_b}{c+\omega_b} = \frac{\cot(\omega_b T/2)-1}{\cot(\omega_b T/2)+1} = \frac{1-\tan(\omega_b T/2)}{1+\tan(\omega_b T/2)}.$$

Figure ?? shows the digital phaser response curves corresponding to the analog response curves in Fig. ??. While the break frequencies are preserved by construction, the notches have moved slightly, although this is not visible from the plots. An overlay of the total phase of the analog and digital allpass chains is shown in Fig. ??. We see that the phase responses of the analog and digital alpass chains diverge visibly only above 9 kHz. The analog phase response approaches zero in the limit as $\omega_a \rightarrow \infty$, while the digital phase response reaches zero at half the sampling rate, 10 kHz in this case.

This is a good example of when the bilinear transform performs very well.



(a) Phase reponses of first-order digital allpass sections with break frequencies at 100, 200, 400, and 800 Hz, with the sampling rate set to 20,000 Hz. (b) Corresponding phaser amplitude response.



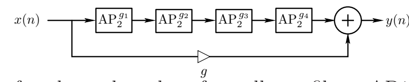
Phase reponse of four first-order allpass sections in series — analog and digital cases overlaid.

Fig. ?? below 10 kHz can be largely eliminated by increasing the sampling rate by 15% or so. See the case of digitizing the Moog VCF for an example in which the presence of feedback in the analog circuit leads to a delay-free loop in the digitized system [?, ?].

Phasing with 2nd-Order Allpass Filters

The allpass structure proposed in [?] provides a convenient means for

generating nonuniformly spaced notches that are independently controllable to a high degree. Another advantage of the allpass approach is that no interpolating delay line is needed.



Structure of a phaser based on four allpass filters AP1 through AP4.

Allpass Phaser Architecture

The architecture of the allpass-based notch filter is shown in Fig. 1. It consists of a series connection of allpass filters with a feed-around. Thus, the delay line of the flanger is replaced by a string of allpass filters. (A delay line is of course an allpass filter itself.) The phaser will have a notch wherever the phase of the allpass chain is at π (180 degrees). It can be shown that these frequencies occur very close to the resonant frequencies of the allpass chain. It is therefore convenient to use a single conjugate pole pair in each allpass section, *i.e.*, use second-order allpass sections of the form

$$H(z) = \frac{a_2 + a_1 z^{-1} + z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

where

$$\begin{aligned} a_1 &= -2R\cos(\theta) \\ a_2 &= R^2 \end{aligned}$$

and R is the radius of each pole in the complex-conjugate pole pair, and pole angles are $\pm\theta$. The pole angle can be interpreted as $\theta = \omega_c T$ where ω_c is the resonant frequency and T is the sampling interval.

Allpass Phaser Parameters

To move just one notch, the tuning of the pole-pair in the corresponding section is all that needs to be changed. Note that tuning affects only one coefficient in the second-order allpass structure (though it is used in two places).

The depth of the notches can be varied together by changing the gain of the feedforward path.

The bandwidth of individual notches is mostly controlled by the distance of the associated pole-pair from the unit circle. So to widen the notch associated with a particular allpass section, one may increase the "damping" of that section.

Finally, since the gain of the allpass string is unity (by definition of allpass filters), the gain of the entire structure is strictly bounded between 0 and 2. This property allows arbitrary notch controls to be applied without fear of the overall gain becoming ill-behaved.

Allpass Phaser Notch Distribution

As mentioned above, it is desirable to avoid exact harmonic spacing of the notches, but what is the ideal non-uniform spacing? One possibility is to space the notches according to the *critical bands of*

hearing, since essentially this gives a uniform notch density with respect to "place" along the basilar membrane in the ear. There is no need to follow closely the critical-band structure, and many simple functional relationships can be utilized to tune the notches [?]. Due to the immediacy of the relation between notch characteristics and the filter coefficients, the notches can easily be placed under musically meaningful control.

Vibrato Simulation

The term *vibrato* refers to small, quasi-periodic variations in the *pitch* of a tone. On a violin, for example, vibrato is produced by wiggling the finger stopping the string on the fingerboard; a violin vibrato frequency can be very slow, or a bit faster than 6 Hz. A typical vibrato *depth* is on the order of 1 percent (a *semitone* is $2^{1/12} \approx 6$ percent). In the singing voice, vibrato is produced by modulating the tension of the vocal folds. Vibrato is typically accompanied by *tremolo*, which is *amplitude modulation* at the same frequency as the vibrato which causes it. For example, in the violin, the frequency-modulations of the string vibrations are translated into amplitude modulations by the complex variations in the frequency response of the violin body.

To apply vibrato to a sound, it is necessary to apply a quasi-periodic *frequency shift*. This can be accomplish using a *modulated delay line*. This works because a time-varying delay line induces a simulated *Doppler shift* on the signal within it.

Doppler Effect

The *Doppler effect* causes the pitch of a sound source to appear to rise or fall due to *motion* of the source and/or listener relative to each other. You have probably heard the pitch of a horn drop lower as it passes by (*e.g.*, from a moving train). As a pitched sound-source moves toward you, the pitch you hear is raised; as it moves away from you, the pitch is lowered. The Doppler effect has been used to enhance the realism of simulated moving sound sources for compositional purposes [?], and it is an important component of the "Leslie effect" (described in §??).

As derived in elementary physics texts, the *Doppler shift* is given by

$$\omega_l = \omega_s \frac{1 + \frac{v_{ls}}{c}}{1 - \frac{v_{sl}}{c}} \tag{6}$$

where ω_s is the radian frequency emitted by the source at rest, ω_l is the frequency received by the listener, v_{ls} denotes the *speed* of the listener relative to the propagation medium in the direction of the source, v_{sl} denotes the speed of the source relative to the propagation medium in the direction of the listener, and c denotes sound speed. Note that all quantities in this formula are scalars.

Vector Formulation

Denote the sound-source *velocity* by $\underline{v}_s(t)$ where t is time. Similarly, let $\underline{v}_l(t)$ denote the velocity of the listener, if any. The *position* of source and listener are denoted $\underline{x}_s(t)$ and $\underline{x}_l(t)$, respectively, where $\underline{x} \stackrel{\Delta}{=} (x_1, x_2, x_3)^T$ is 3D position. We have velocity related to position by

$$\underline{v}_s = \frac{d}{dt}\underline{x}_s(t) \qquad \underline{v}_l = \frac{d}{dt}\underline{x}_l(t). \tag{7}$$

Consider a Fourier component of the source at frequency ω_s. We wish to know how this frequency is shifted to ω_l at the listener due to the Doppler effect.

Velocity Projection

The Doppler effect depends only on velocity components along the line connecting the source and listener [?, p. 453]. We may therefore *orthogonally project* the source and listener velocities onto the vector $\underline{x}_{sl} = \underline{x}_l - \underline{x}_s$ pointing from the source to the listener. (See Fig. 1.1 on page 44 for a specific example.)

The *orthogonal projection* of a vector $\underline{x}$ onto a vector $\underline{y}$ is given by [?]

$$\mathcal{P}_{\underline{y}}(\underline{x}) = \frac{\langle \underline{x}, \underline{y} \rangle}{\|\, \underline{y} \,\|^2}\underline{y} \stackrel{\Delta}{=} \frac{\underline{x}^T \underline{y}}{\underline{y}^T \underline{y}}\underline{y}.$$

Therefore, we can write the projected source velocity as

$$\underline{v}_{sl} = \mathcal{P}_{\underline{x}_{sl}}(\underline{v}_s) = \frac{\langle \underline{v}_s, \underline{x}_{sl} \rangle}{\|\, \underline{x}_{sl} \,\|^2}\underline{x}_{sl} = \frac{\langle \underline{v}_s, \underline{x}_l - \underline{x}_s \rangle}{\|\, \underline{x}_l - \underline{x}_s \,\|^2}(\underline{x}_l - \underline{x}_s). \tag{8}$$

In the *far field* (listener far away), Eq. (8) reduces to

$$\underline{v}_{sl} \approx \frac{\langle \underline{v}_s, \underline{x}_l \rangle}{\|\, \underline{x}_l \,\|^2}\underline{x}_l = \mathcal{P}_{\underline{x}_l}(\underline{v}_s) \qquad (\|\, \underline{x}_l \,\| \gg \|\, \underline{x}_s \,\|). \tag{9}$$

1.1 Doppler Simulation

It is well known that a time-varying delay line results in a frequency shift. Time-varying delay is often used, for example, to provide *vibrato* and *chorus* effects [?]. We therefore expect a time-varying

delay-line to be capable of precise Doppler simulation. This section discusses simulating the Doppler effect using a variable delay line [?].

Consider Doppler shift from a physical point of view. The air can be considered as analogous to a *magnetic tape* which moves from source to listener at speed c. The source is analogous to the *write-head* of a tape recorder, and the listener corresponds to the *read-head*. When the source and listener are fixed, the listener receives what the source records. When either moves, a Doppler shift is observed by the listener, according to Eq. (6).

Doppler Simulation via Delay Lines

This analogy also works for a delay-line based computational model. The magnetic tape is now the delay line, the tape read-head is the read-pointer of the delay line, and the write-head is the delay-line write-pointer. In this analogy, it is readily verified that modulating delay by changing the read-pointer increment from 1 to $1 + v_{ls}/c$ (thereby requiring interpolated reads) corresponds to listener motion away from the source at speed v_{ls}. It also follows that changing the *write-pointer* increment from 1 to $1 + v_{sl}/c$ corresponds *source motion toward the listener* at speed v_{sl}. When this is done, we must use *interpolating writes* into the delay memory. Interpolating writes are often called *de-interpolation* [?], and they are formally the graph-theoretic *transpose* of interpolating reads (ordinary "interpolation") [?]. A review of time-varying, interpolating, delay-line reads and writes, together with a method using a single shared pointer, are given in [?].

Time-Varying Delay-Line Reads

If $x(t)$ denotes the input to a time-varying delay, the output can be written as

$$y(t) = x(t - D_t).$$

where D_t denotes the time-varying delay in seconds. In discrete-time implementations, when D_t is not an integer multiple of the sampling interval, $x(t - D_t)$ may be approximated to arbitrary accuracy (in a finite band) using *bandlimited interpolation* (see §?? on page ??) or other techniques for implementation of *fractional delay* [?, ?].

Let's analyze the frequency shift caused by a time-varying delay by setting $x(t)$ to a complex sinusoid at frequency ω_s:

$$x(t) = e^{j\omega_s t}$$

The output is now

$$y(t) = x(t - D_t) = e^{j\omega_s \cdot (t - D_t)}.$$

The instantaneous phase of this signal is

$$\theta(t) = \angle y(t) = \omega_s \cdot (t - D_t)$$

which can be differentiated to give the instantaneous frequency

$$\omega_l = \omega_s(1 - \dot{D}_t) \tag{10}$$

where ω_l denotes the output frequency, and $\dot{D}_t \stackrel{\Delta}{=} \frac{d}{dt}D_t$ denotes the time derivative of the delay D_t. Thus, *the delay growth-rate*, $\dot{D}_t$, equals the *relative frequency downshift*:

$$\dot{D}_t = \frac{\omega_s - \omega_l}{\omega_s}.$$

Comparing Eq. (10) to Eq. (6), we find that the time-varying delay most naturally simulates Doppler shift caused by a *moving listener*, with

$$\dot{D}_t = -\frac{v_{ls}}{c}. \tag{11}$$

That is, the delay growth-rate, $\dot{D}_t$, should be set to the speed of the listener *away* from the source, normalized by sound speed c.

Simulating *source* motion is also possible, but the relation between delay change and desired frequency shift is more complex, *viz.*, from Eq. (6) and Eq. (10),

$$\dot{D}_t = -\frac{\frac{v_{ls}}{c} + \frac{v_{sl}}{c}}{1 - \frac{v_{sl}}{c}} \approx -\left(\frac{v_{ls}}{c} + \frac{v_{sl}}{c}\right)$$

where the approximation is valid for $v_{sl} \ll c$. In Section 1.1, a simplified approach is proposed based on moving the delay *input* instead of its output.

The time-varying delay line was described in §**??** on page **??**. As discussed there, to implement a *continuously* varying delay, we add a "delay growth parameter" g to the `delayline` function in Fig. **??** on page **??**, and change the line

```
rptr += 1; // pointer update
```

to

```
rptr += 1 - g; // pointer update
```

When g is 0, we have a fixed delay line, corresponding to $\dot{D}_t = 0$ in Eq. (10). When g > 0, the delay grows g samples per sample, which

we may also interpret as seconds per second, *i.e.*, $\dot{D}_t = \mathrm{g}$. By Eq. (11), we see that we need

$$\mathrm{g} = -\frac{v_{ls}}{c}$$

to simulate a listener traveling toward the source at speed v_{ls}.

Note that when the read- and write-pointers are driven directly from a model of physical propagation-path geometry, they are always separated by predictable minimum and maximum delay intervals. This implies it is unnecessary to worry about the read-pointer *passing* the write-pointers or vice versa. In generic *frequency shifters* [**?**], or in a Doppler simulator not driven by a changing geometry, a pointer cross-fade scheme may be necessary when the read- and write-pointers get too close to each other.

Multiple Read Pointers

Using *multiple read pointers*, multiple listeners can be simulated. Furthermore, each read-pointer signal can be *filtered* to simulate propagation losses and radiation characteristics of the source in the direction of the listener. The read-pointers can move independently to simulate the different Doppler shifts associated with different listener motions and relative source directions.

Multiple Write Pointers

It is interesting to consider also what effects can be achieved using multiple de-interpolating write pointers. From the considerations in

§1.1, we see that multiple write-pointers correspond to multiple write-heads on a magnetic tape recorder. If they are arranged at a fixed spacing, they are equivalent to multiple read pointers, providing a basic multipath simulation. If, however, the write pointers are moving *independently*, they induce *independent Doppler shifts due to source motion*. In particular, each write-pointer can lay down a signal from a separate source to a single listener with its own Doppler shift. Furthermore, each write-signal can be passed through its own filter. Such an individualized source filter can implement all filtering incurred along the propagation path from each source to the listener.

When all write pointers have the same input signal, their filters can be implemented using a *series chain* in which the outputs of successive filters in the chain correspond to progressively longer propagation paths (progressively more filtering). Such an implementation can greatly reduce the filter order required for propagation paths longer than the shortest.

The write-pointers may cross each other with no ill effects, since all but the first[8] simply sum into the shared delay line.

We have seen that a single delay line can be used to simulate any number of moving listeners (§1.1) or any number of moving sources. However, when simulating *both* multiple listeners and multiple sources, it is not possible to share a single delay line. This is because the different listeners do not see the same Doppler shift for each moving source, and while the listener's read-pointer motion can be adjusted to correct for the Doppler shift seen from any particular source, it cannot correct for more than one in general. Thus, in

[8]The "first" write-pointer is defined as the one writing farthest ahead in time; it must *overwrite* memory, instead of summing into it, when a circular buffer is being used, as is typical.

general, we need as many delay lines as there are sources or listeners, whichever is smaller. More precisely, if there are N moving sources and M moving listeners, simulation requires $\min(N, M)$ delay lines.

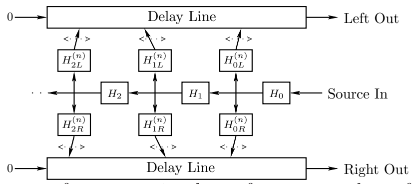
Stereo Processing

As a special case, stereo processing of any number of sources can be accomplished using two delay lines, corresponding to left and right stereo channels. The stereo mix may contain a panned mixture of any number sources, each with its own stereo placement, path filtering, and Doppler shift. The two stereo outputs may correspond to "left and right ears," or, more generally, to left- and right-channel microphones in a studio recording set-up.

System Block Diagram

A schematic diagram of a stereo multiple-source simulation is shown in Fig. 1.1. To simplify the layout, the input and output signals are all on the right in the diagram. For further simplicity, only one input source is shown. Additional input sources are handled identically, summing into the same delay lines in the same way.

The input source signal first passes through filter $H_0(z)$, which provides *time-invariant* filtering common to all propagation paths. The left- and right-channel filters $H_{0L}^{(n)}(z)$ and $H_{0R}^{(n)}(z)$ are typically low-order, linear, *time-varying* filters implementing the time-varying



Block diagram of a stereo simulator for any number of moving sound sources (from [?]).

characteristics of the shortest (time-varying) propagation path from the source to each listener. (The (n) superscript here indicates a time-varying filter.) These filter outputs *sum* into the delay lines at arbitrary (time-varying) locations using interpolating writes (de-interpolation). The zero signals entering each delay line on the left can be omitted if the left-most filter overwrites delay memory instead of summing into it.

The outputs of $H_{0L}^{(n)}(z)$ and $H_{0R}^{(n)}(z)$ in Fig. 1.1 correspond to the "direct signal" from the moving source, when a direct signal exists. These filters may incorporate modulation of losses due to the changing propagation distance from the moving source to each listener, and they may include dynamic equalization corresponding to the changing radiation strength in different directions from the moving (and possibly turning) source toward each listener.

The next trio of filters in Fig. 1.1, $H_1(z)$, $H_{1L}^{(n)}(z)$, and $H_{1R}^{(n)}(z)$, correspond to the next-to-shortest acoustic propagation path, typically the "first reflection," such as from a wall close to the source. Since a reflection path is longer than the direct path, and since a reflection itself can attenuate (or scatter) an incident sound ray,

there is generally more filtering required relative to the direct signal. This additional filtering can be decomposed into its fixed component $H_1(z)$ and time-varying components $H_{1L}^{(n)}(z)$ and $H_{1R}^{(n)}(z)$.

Note that acceptable results may be obtained without implementing all of the filters indicated in Fig. 1.1. Furthermore, it can be convenient to incorporate $H_i(z)$ into $H_{iL}^{(n)}(z)$ and $H_{iR}^{(n)}(z)$ when doing so does not increase their orders significantly.

Note also that the source-filters $H_{iL}^{(n)}(z)$ and $H_{iR}^{(n)}(z)$ may include HRTF filtering [?, ?] in order to impart illusory angles of arrival in 3D space.

Chorus Effect

The *chorus effect* (or "choralizer") is any signal processor which makes one sound source (such as a voice) sound like *many* such sources singing (or playing) in *unison*. Since performance in unison is never exact, chorus effects simulate this by making independently modified copies of the input signal. Modifications may include

(1) delay,

(2) frequency shift, and

(3) amplitude modulation.

The typical chorus effect today is based on several *time-varying delay lines* which accomplishes (1) and (2) in a qualitative fashion. Reverb generally provides (3) incidentally. Before digital delay lines, analog

LC ladder networks were used as an approximation, beginning in the early 1940s in the Hammond organ [?, p. 731].

An efficient chorus-effect implementation may be based on *multiple interpolating taps* working on a single delay line. The taps oscillate back and forth about the positions they would have while implementing a fixed tapped delay line. The tap modulation frequency may be set to achieve a prescribed frequency-shift via the Doppler effect. Each tap should be individually spatialized; in the case of stereo, each tap can be panned to its own stereo position.

The Leslie

The *Leslie*, named after its inventor, Don Leslie,[9] is a popular audio processor used with electronic organs and other instruments [?, ?]. It employs a rotating horn and rotating speaker port to "choralize" the sound. Since the horn rotates within a cabinet, the listener hears multiple reflections at different Doppler shifts, giving a kind of *chorus effect*. Additionally, the Leslie amplifier distorts at high volumes, producing a pleasing "growl" highly prized by keyboard players.

The Leslie consists primarily of a rotating horn and a rotating speaker port inside a wooden cabinet enclosure [?]. We first consider the rotating horn.

Rotating Horn Simulation

The heart of the Leslie effect is a rotating horn loudspeaker. The rotating horn from a Model 600 Leslie can be seen mounted on a

[9] http://en.wikipedia.org/wiki/Leslie_speaker

microphone stand in Fig. ??. Two horns are apparent, but one is a dummy, serving mainly to cancel the centrifugal force of the other during rotation. The Model 44W horn is identical to that of the Model 600, and evidently standard across all Leslie models [?]. For a circularly rotating horn, the source position can be approximated as

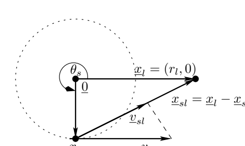
$$\underline{x}_s(t) = \begin{bmatrix} r_s\cos(\omega_m t) \\ r_s\sin(\omega_m t) \end{bmatrix} \qquad (12)$$

where r_s is the circular radius and ω_m is angular velocity. This expression ignores any *directionality* of the horn radiation, and approximates the horn as an omnidirectional radiator located at the same radius for all frequencies. In the Leslie, a *diffuser* is inserted into the end of the horn in order to make the radiation pattern closer to uniform [?], so the omnidirectional assumption is reasonably accurate.

By Eq. (7), the source velocity for the circularly rotating horn is

$$\underline{v}_s(t) = \frac{d}{dt}\underline{x}_s(t) = \begin{bmatrix} -r_s\omega_m\sin(\omega_m t) \\ r_s\omega_m\cos(\omega_m t) \end{bmatrix} \qquad (13)$$

Note that the source velocity vector is always orthogonal to the source position vector, as indicated in Fig. 1.1.



Relevant geometry for a rotating horn (from [?]).

Since $\underline{v}_s$ and $\underline{x}_s$ are orthogonal, the projected source velocity Eq. (8) simplifies to

$$\underline{v}_{sl} = \mathcal{P}_{\underline{x}_{sl}}(\underline{v}_s) = \frac{\langle \underline{v}_s, \underline{x}_l \rangle}{\| \underline{x}_l - \underline{x}_s \|^2} (\underline{x}_l - \underline{x}_s) . \qquad (14)$$

Arbitrarily choosing $\underline{x}_l = (r_l, 0)$ (see Fig. 1.1), and substituting Eq. (12) and Eq. (13) into Eq. (14) yields

$$\underline{v}_{sl} = \frac{-r_l r_s \omega_m \sin(\omega_m t)}{r_l^2 + 2 r_l r_s \cos(\omega_m t) + r_s^2} \begin{bmatrix} r_l - r_s \cos(\omega_m t) \\ -r_s \sin(\omega_m) t \end{bmatrix} . \qquad (15)$$

In the far field, this reduces simply to

$$\underline{v}_{sl} \approx -r_s \omega_m \sin(\omega_m t) \begin{bmatrix} 1 \\ 0 \end{bmatrix} . \qquad (16)$$

Substituting into the Doppler expression Eq. (6) with the listener velocity v_l set to zero yields

$$\omega_l = \frac{\omega_s}{1 + r_s \omega_m \sin(\omega_m t)/c} \approx \omega_s \left[1 - \frac{r_s \omega_m}{c} \sin(\omega_m t) \right] , \qquad (17)$$

where the approximation is valid for small Doppler shifts. Thus, in the far field, a rotating horn causes an approximately *sinusoidal* multiplicative frequency shift, with the amplitude given by horn length r_s times horn angular velocity ω_m divided by sound speed c. Note that $r_s \omega_m$ is the *tangential speed* of the assumed point of horn radiation.

Leslie Free-Field Horn Measurements

The free-field radiation pattern of a Model 600 Leslie rotating horn was measured using the experimental set-up shown in Fig. ?? [?]. A

matched pair of Panasonic microphone elements (Crystal River Snapshot system) were used to measure the horn response both in the plane of rotation and along the axis of rotation (where no Doppler shift or radiation pattern variation is expected). The microphones were mounted on separate boom microphone stands, as shown in the figure. A close-up of the plane-of-rotation mic is shown in Fig. ??.

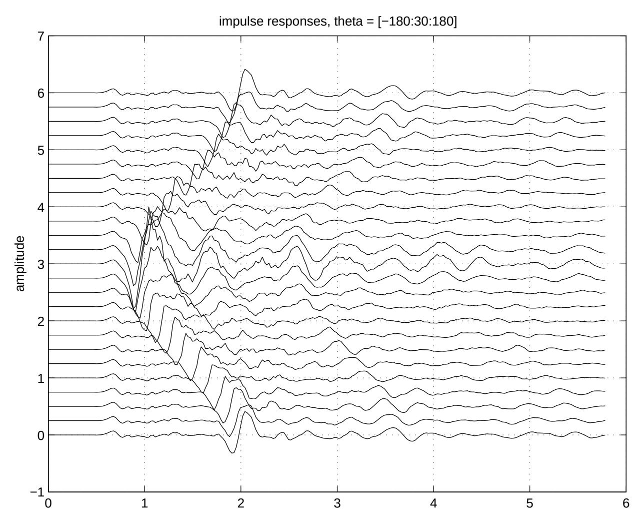
Rotating horn recording set up (from [?]).

Microphone close-up (from [?]).

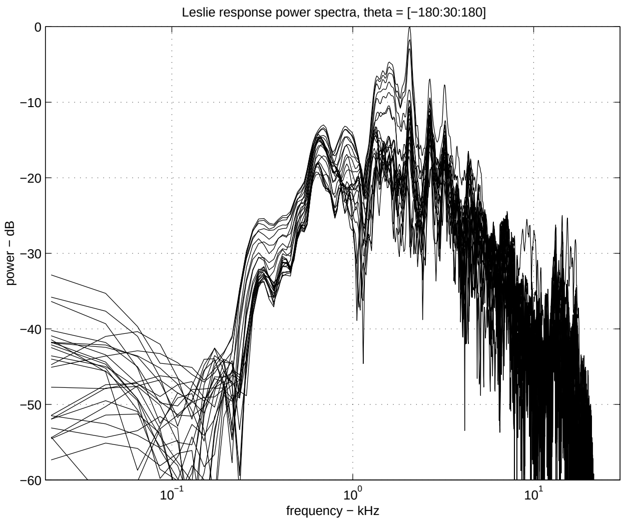
The horn was set manually to fixed angles from -180 to 180 degrees in increments of 15 degrees, and at each angle the impulse response was measured using 2048-long Golay-code pairs [?].

Figure ?? shows the measured impulse responses and Fig. ?? shows the corresponding amplitude responses at the various angles. Note that the beginning of each impulse response contains a fixed portion which does not depend significantly on the angle. This is thought to be due to “leakage” from the base of the horn. It arrives first since

the straight-line path from the enclosed speaker to the microphone is shorter than that traveling through the horn assembly.



Measured impulse-responses of the Leslie 600 rotating-horn at multiples of 15 degrees. The middle trace is recorded with the microphone along the axis of the horn (from [?]).



Measured amplitude-responses of the Leslie 600 rotating-horn at multiples of 15 degrees (from [?]).

Separating Horn Output from Base Leakage

Note that Fig. ?? indicates the existence of fixed and angle-dependent components in the measured impulse responses. An iterative algorithm was developed to model the two components separately [?].

Let $M = 256$ denote the number of impulse-response samples in

each measured impulse response,and let $N = 25$ denote the number of angles (-180:15:180) at which impulse-response measurements were taken. We denote the $M \times N$ impulse-response matrix by $\mathbf{h}$. Each column of $\mathbf{h}$ is an impulse response at some horn angle. (Figure ?? can be interpreted as a plot of the *transpose* of $\mathbf{h}$.)

We model $\mathbf{h}$ as

$$\mathbf{h} = \boldsymbol{\alpha} + \boldsymbol{\gamma} \cdot \text{diag}(z^{-\tau_i}) + \mathbf{e}$$

where τ_i is the arrival-time delay, in samples, for the horn output in the ith row (the delays clearly visible in Fig. ?? as a function of angle). These arrival times are estimated as the location of the peak in the cross-correlation between the ith impulse response and the same impulse response after converting it to minimum phase [?]. The diagonal matrix diag$(z^{-\tau_i})$ denotes a *shift operator* which delays the ith column of $\boldsymbol{\gamma}$ by τ_i samples. Thus, $\boldsymbol{\gamma}$ contains the horn-output impulse response (without the base leakage) shifted to time zero (*i.e.*, the angle-dependent delay is removed). Finally, the error matrix $\mathbf{e}$ is to be minimized in the least-squares sense.

Each column of the matrix $\boldsymbol{\alpha}$ contains a copy of the estimated horn-base leakage impulse-response:

$$\boldsymbol{\alpha} = \underline{a} \cdot \mathbf{1}^T$$

where $\mathbf{1}^T = [1, 1, \ldots, 1]$.

The estimated angle-dependent impulse-responses in $\boldsymbol{\gamma}$ are modeled as linear combinations of $K = 5$ *fixed* impulse responses, viewed (loosely) as *principal components*:

$$\boldsymbol{\gamma} = \mathbf{g} \cdot \mathbf{w}$$

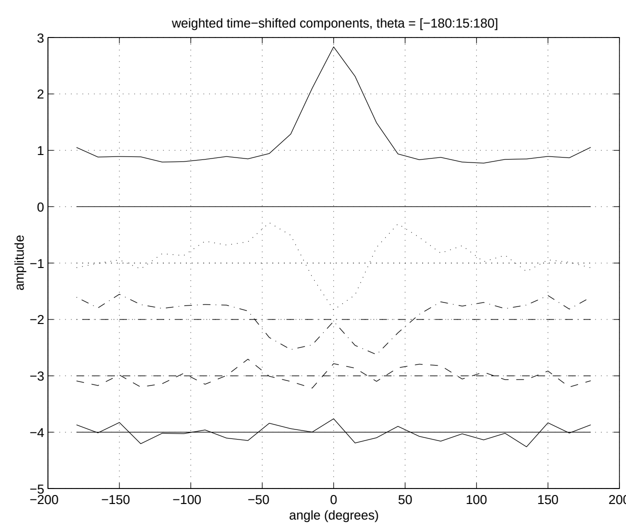
where $\mathbf{g}$ is the $M \times K$ orthonormal matrix of fixed filters (principal components), and $\mathbf{w}$ is a $K \times N$ matrix of *weights*, found in the usual way by a truncated *singular value decomposition* (SVD) [?].

Algorithm

To start the separation algorithm, $\boldsymbol{\gamma}_0$ is initialized to the zero-shifted impulse response data $\mathbf{h} \cdot \text{diag}(z^{\tau_i})$, ignoring the tails of the base-leakage they may contain. Then $\boldsymbol{\alpha}_0$ is estimated as the mean of $\mathbf{h} - \boldsymbol{\gamma}_0\text{diag}(z^{-\tau_i})$. This mean is then subtracted from $\mathbf{h}$ to produce $\mathbf{b}_1 = (\mathbf{h} - \boldsymbol{\alpha}_0)\text{diag}(z^{-\tau_i})$ which is then then converted to $\boldsymbol{\gamma}_1 = \mathbf{g}_1 \cdot \mathbf{w}_1$ by a truncated SVD. A revised base-leakage estimate $\boldsymbol{\alpha}_1$ is then formed as $\mathbf{h} - \boldsymbol{\gamma}_1\text{diag}(z^{-\tau_i})$, and so on, until convergence is achieved.

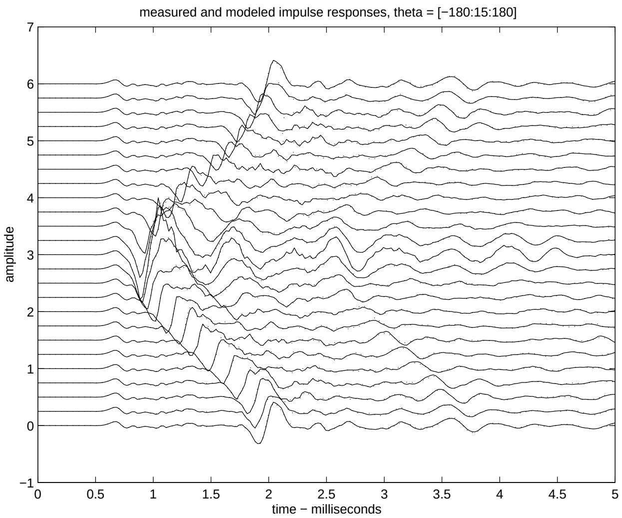
Results

Figure ?? plots the $K = 5$ weighted principal components identified for the angle-dependent component of the horn radiativity. Each component is weighted by its corresponding singular value, thus visually indicating its importance. Also plotted using the same line type are the zero-lines for each principal component. Note in particular that the first (largest) principal component is entirely positive.



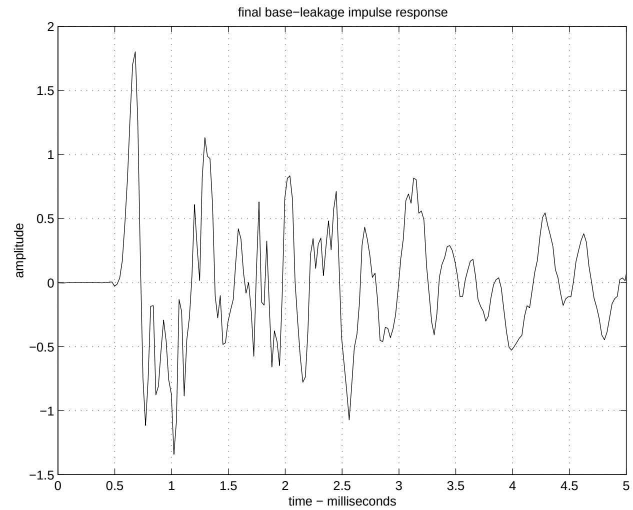
First 5 principal components weighted by their corresponding singular values. Each angle-dependent impulse response is modeled as a linear combination of these angle-independent impulse-response components (from [?]).

Figure ?? shows the complete horn impulse-response model $(\boldsymbol{\alpha} + \boldsymbol{\gamma} \cdot \text{diag}(z^{-\tau_i}))$, overlaid with the original raw data $\mathbf{h}$. We see that both the fixed base-leakage and the angle-dependent horn-output response are closely followed by the fitted model.

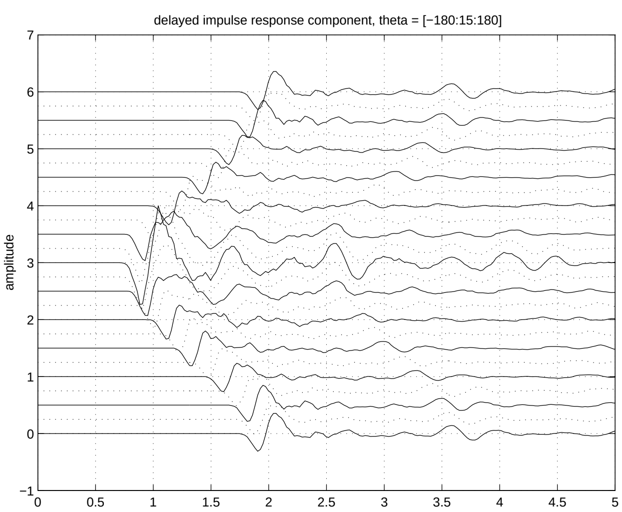


Overlay of measured (solid) and modeled (dotted) impulse-responses at multiples of 15 degrees (from [?]).

Figure ?? shows the estimated impulse response of the base-leakage component $\underline{a}(n)$, and Fig. ?? shows the modeled angle-dependent horn-output components γ delayed out to their natural arrival times.



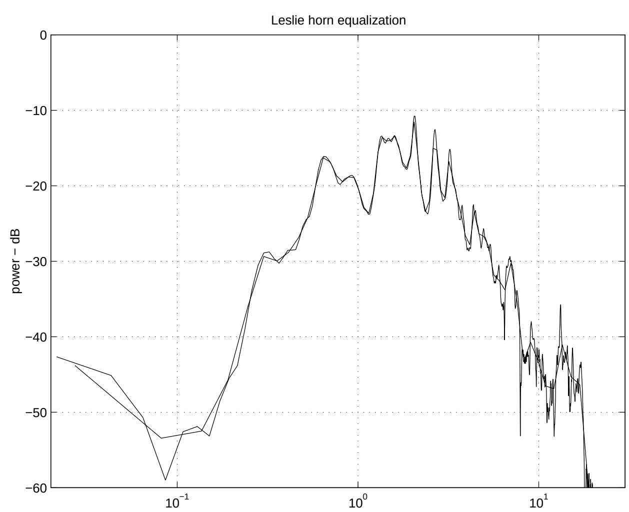
Modeled base-leakage impulse-response (angle-independent) (from [?]).



Modeled horn-output impulse-responses at multiples of 15 degrees (from [?]).

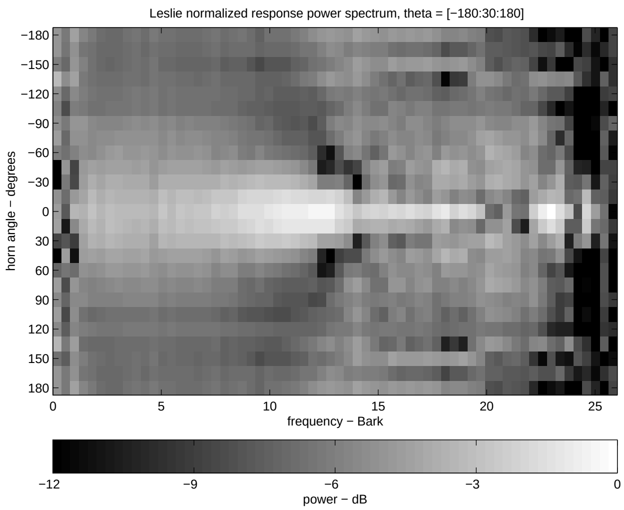
Figure ?? shows the average power response of the horn outputs. Also overlaid in that figure is the average response smoothed according to Bark frequency resolution [?]. This equalizer then becomes $H_0(z)$ in Fig. 1.1. The filters $H_{0L}(z)$ and $H_{0R}(z)$ in Fig. 1.1 are obtained by dividing the Bark-smoothed frequency-response at each angle by $H_0(z)$ and designing a low-order recursive filter to provide that equalization dynamically as a function of horn angle.

The impulse-response arrival times τ_i determine where in the delay lines the filter-outputs are to be summed in Fig. 1.1.



Average angle-dependent amplitude response overlaid with Bark-smoothed response to be used as a fixed equalization applied to the source (from [?]).

Figure ?? shows a spectrogram view of the angle-dependent amplitude responses of the horn with $H_0(z)$ (Bark-smoothed curve in Fig. ??) divided out. This angle-dependent, differential equalization is used to design the filters $H_{0L}(z)$ and $H_{0R}(z)$ in Fig. 1.1. Note that below 12 Barks or so, the angle-dependence is primarily to decrease amplitude as the horn points away from the listener, with high frequencies decreasing somewhat faster with angle than low frequencies.



Angle-dependent amplitude response divided by Bark-smoothed average response to be used as the basis for design of time-varying, angle-dependent equalization to be applied after $H_0(z)$ (from [?]).

1.2 Rotating Woofer-Port and Cabinet Simulation

It is straightforward to extend our computational model to include the rotating woofer port and wooden cabinet enclosure as follows:

- In [?], it is mentioned that an AM "throb" is the main effect of the rotating woofer port. A modulated lowpass-filter cut-off frequency has been used for this purpose by others. Our measured data will be used to construct angle-dependent filtering in a manner analogous to that of the rotating horn, and this "woofer filter" runs in parallel with the rotating horn model.
- The Leslie cabinet multiply-reflects the sound emanating from the rotating horn. The first few early reflections are simply handled as additional sources in Fig. 1.1. We can extend the impulse-response-component separation algorithm of §1.1 to the case of superimposed early reflections in the impulse response. (Preliminary results are promising.)
- To qualitatively simulate later, more *reverberant* reflections in the Leslie cabinet, we feed a portion of the rotating-horn and speaker-port signals to separate states of an *artificial reverberator* (see Chapter ??). This reverberator may be configured as a "very small room" corresponding to the dimensions and scattering characteristics of the Leslie cabinet, and details of the response may be calibrated using measurements of the impulse response of the Leslie cabinet. Finally, in order to emulate the natural spatial diversity of a radiating Leslie cabinet in a room, "virtual cabinet vent outputs" can be extracted from the model and fed into separate states of a *room reverberator*.

In summary, we may use multiple interpolating write-pointers to individually simulate the early cabinet reflections, and a "Leslie cabinet" reverberator for handling later reflections more statistically.

1.3 Miscellaneous Effects

This section describes miscellaneous digital audio effects which the author has seen applied in practice. For much more about signal processing for digital audio effects, see, *e.g.*, [?].

Doubling Simulation

Doubling is a studio recording technique often used to "thicken" vocals in which the same part is sung twice by the same person. In other words, doubling is a "chorus of two", where both parts are sung "in unison" by the same person. As an example, the Beatles used doubling very often, such as on the track "Hard Day's Night". A single variable delay line can simulate doubling very effectively.

Slap Back

The term *slap back* refers to the use of a single echo on a recorded track. The echo may be placed in a different spatial location in the stereo mix. Normally the echo delay is just large enough to be heard as a discrete echo on careful listening (*e.g.*, on the order of tens of milliseconds). Slap back is very popular in 1950s-style recordings such as "rockabilly" tunes.

In summary, slap back can be regarded as a simplification of doubling in which the second voice is kept at a larger, fixed delay relative to the first voice.