12

Virtual analog effects

V. Välimäki, S. Bilbao, J. O. Smith, J. S. Abel, J. Pakarinen and D. Berners

12.1 Introduction

Virtual analog effects are a consequence of the ongoing digitization of all equipment used in music production. Various digital methods to imitate the warm or lo-fi sound qualities that remind listeners of analog times are covered in this chapter. In particular, many algorithms presented in this chapter are physical models of audio effect boxes that have been traditionally analog electronic or electromechanical devices, such as voltage-controlled filters and spring reverberation units. Some algorithms, for instance the telephone sound effect, are signal models, which produce analog-sounding results without analog circuit analysis. Almost all algorithms are nonlinear and produce distortion. A few virtual analog effects, such as the wah-wah filter, phase, and vintage valve amplifier simulation, are also mentioned elsewhere in this book.

The following virtual analog effect processing techniques are reviewed in this chapter. Section 12.2 discusses virtual analog filters. We start with second- and higher-order filters, which include nonlinear elements, and proceed to equalizers and filter-based effect-processing algorithms. Models for valve amplifiers are reviewed in Section 12.3. Simulation of spring and plate reverberations units is described in Section 12.4. Section 12.5 focuses on simulation of vintage tape-based echo units. Finally, Section 12.6 gives an introduction to audio antiquing, which refers to the transformation of clean, modern audio files into ancient-sounding analog recordings.

12.2 Virtual analog filters

12.2.1 Nonlinear resonator

Analog filters used in music technology are not strictly speaking linear, because at high signal levels they produce distortion. One attempt to include this phenomenon in digital filters has been

DAFX: Digital Audio Effects, Second Edition. Edited by Udo Zölzer.

described by Rossum [Ros92]. He proposed inserting a saturating nonlinearity in the feedback path of a second-order all-pole filter, see Figure 12.1. In this case, the clipper is a symmetrical hard limiter. With this modification, the filter behaves a lot like an analog filter. It produces harmonic distortion and compression when the input signal level is high. Furthermore, its resonance frequency changes when the filter is overloaded, as pointed out in [Ros92]. This technique was used in the E-mu EMAX II sampler, which appeared in 1989. It is an early example of a virtual analog filter used in commercial products.

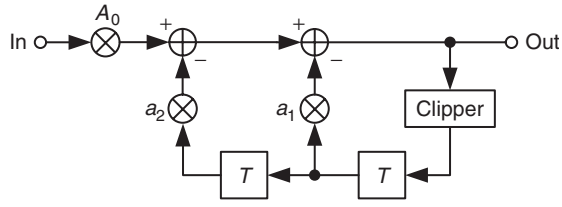

Figure 12.1 A digital resonant filter with a nonlinear element in its feedback path [Ros92].

A MATLAB® implementation of Rossum's nonlinear resonator is given in M-file 12.1. The filter coefficients A_0, a_1, and a_2 are computed as in a conventional digital resonator, see, e.g., [Ste96]. In the following examples, we set the resonance frequency to 1 kHz and the bandwidth to 20 Hz. The sampling rate is $f_S = 44.1$ kHz. The saturation limit of the clipper is set to 1.0.

When the amplitude of the input signal is 1.0, no distortion occurs, because the signal level at the input of the first state variable (unit delay) does not exceed the saturation level. However, when the input amplitude is larger than the saturation limit, the filter compresses and distorts the signal. At the same time aliasing occurs, since all distortion components above the Nyquist limit get mirrored back to the audio band. To avoid this, it would be necessary run the filter at a higher sample rate.

Figure 12.2 shows two examples of the power spectrum of the output signal of the nonlinear resonator, when the input signal is a white-noise sequence. In Figure 12.2(a) the random samples are uniformly distributed between −1.0 and 1.0, while in Figure 11.2(b) they are distributed between −40 and 40. Again, the limit has been set to 1.0. In Figure 12.2(a) the peak in the power spectrum appears at 1000 Hz, as expected, and the −3 dB bandwidth is 20 Hz. However, in Figure 12.2(b), the center frequency has changed: It is now about 1030 Hz. Additionally, the bandwidth is wider than before, about 70 Hz. These phenomena are caused by the saturating nonlinear element, as explained in [Ros92].

M-file 12.1 (nlreson.m)

```
function y = nlreson(fc, bw, limit, x)
% function y = nlreson(fc, bw, limit, x)
% Authors: Välimäki, Bilbao, Smith, Abel, Pakarinen, Berners
%
% Parameters:
% fc = center frequency (in Hz)
% bw = bandwidth of resonance (in Hz)
% limit = saturation level
% x = input signal
fs = 44100;  % Sampling rate
R = 1 - pi*(bw/fs);  % Calculate pole radius
costheta = ((1+(R*R))/(2*R))*cos(2*pi*(fc/fs)); % Cosine of pole angle
a1 = -2*R*costheta;  % Calculate first filter coefficient
a2 = R*R;  % Calculate second filter coefficient
```

```
A0 = (1 - R*R)*sin(2*pi*(fc/fs)); % Scale factor
y = zeros(size(x));  % Allocate memory for output signal
w1 = 0; w2 = 0;  % Initialize state variables (unit delays)
y(1) = A0*x(1);  % The first input sample goes right through
w0 = y(1);  % Input to the saturating nonlinearity
for n = 2:length(x);  % Process the rest of input samples
    if y(n-1) > limit, w0 = limit;  % Saturate above limit
    elseif y(n-1) < -limit, w0 = - limit;  % Saturate below limit
    else w0 = y(n-1);end  % Otherwise do nothing
    w2 = w1;  % Update the second state variable
    w1 = w0;  % Update the first state variable
    y(n) = A0*x(n) - a1*w1 - a2*w2;  % Compute filter output
end
```

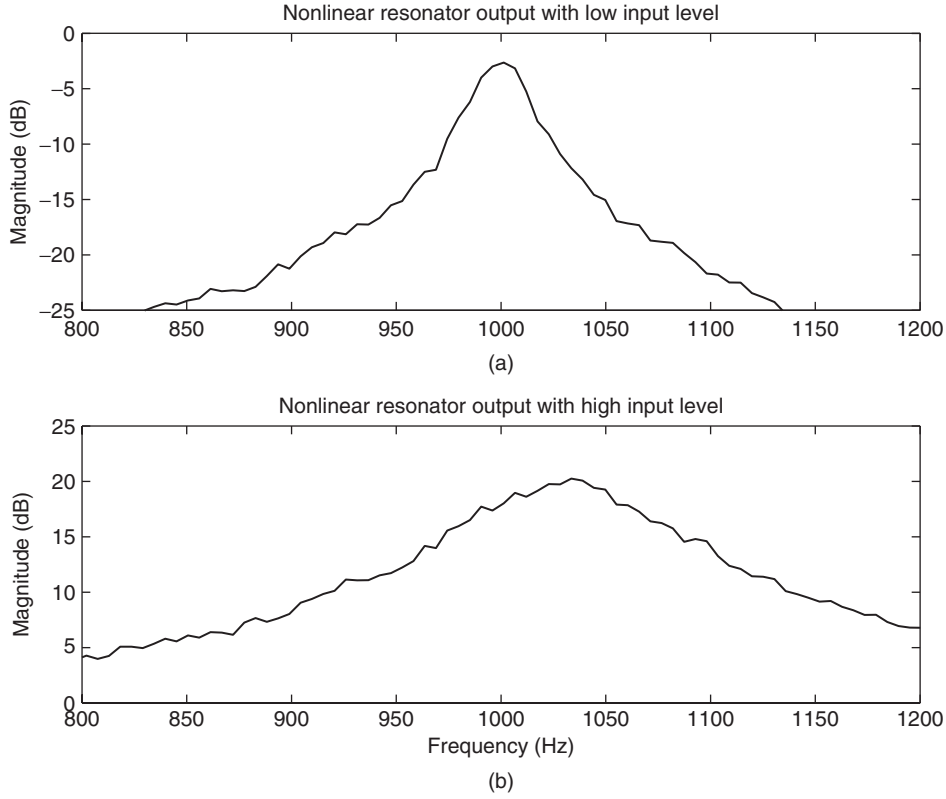

Figure 12.2 Power spectrum of the output signal when the input signal is (a) a low-level and (b) a high-level white-noise sequence.

12.2.2 Linear and nonlinear digital models of the Moog ladder filter

Next we discuss a well-known analog filter originally proposed by Robert Moog for his synthesizer designs [Moo65]. The filter consists of four one-pole filters in cascade and a global negative feedback loop. The structure is called a ladder filter because of the way in which the one-pole filter sections are cascaded. The filter includes a feedback gain factor k, which may be varied between

0 and 4. This unusual choice of range comes from the fact that at the cut-off frequency the gain of each filter section is 1/4. When $k = 4.0$, the Moog filter oscillates at its cut-off frequency (i.e., self-oscillates). The Moog ladder filter is also famous for its uncoupled control of the resonance (Q value) and the corner frequency, which are directly adjusted by the feedback gain k and the cut-off frequency parameters, respectively.

Stilson and Smith [SS96] have considered various methods of converting the Moog ladder filter into a corresponding digital filter. They found that standard transforms, the bilinear and the backward difference transform, yield a structure containing a delay-free path from input to output. To solve this problem, an ad-hoc unit delay may be inserted into the feedback loop. Figure 12.3 shows the resulting filter structure with the extra unit delay. Since the one-pole filters remain first-order filters in the digital implementation, but there is now an extra unit delay, the overall system becomes a fifth-order filter.

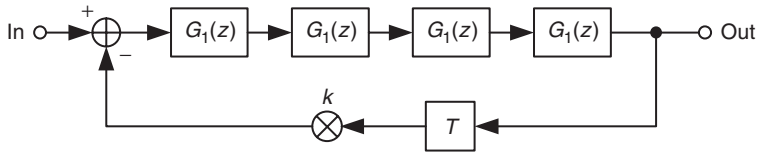

Figure 12.3 The digital Moog filter structure proposed in [SS96].

Another complication in the discretization of the Moog filter is that the constant-Q control of the corner frequency is lost. This happens with both the bilinear and the backward difference transform, which place the zeros of the first-order filter at different locations. Stilson and Smith have found a useful compromise first-order filter that has largely independent control of the Q value with corner frequency:

$$G_1(z) = \frac{1+p}{1.3}\frac{1+0.3z^{-1}}{1+pz^{-1}}, \tag{12.1}$$

where $0 \leq p \leq 1$ is the pole location. It can be determined conveniently from the normalized cut-off frequency f_c (in Hz),

$$p = \frac{f_c}{f_S}. \tag{12.2}$$

However, this relation is accurate only when the cut-off frequency is not very high (below 2 kHz or so), and otherwise the actual corner frequency of the filter will be higher than expected. A fourth-order polynomial correction to the pole location has been presented in [VH06]. The Q value of the digital Moog filter proposed by Stilson and Smith [SS96] also deviates slightly from constant at large Q values, but this error may be negligible, since humans are not very sensitive to variations in Q [SS96].

Several alternative versions of the digital Moog filter have been proposed over the years. Wise has shown that a cascade of digital allpass filters can be used to realize a filter with a similar behavior [Wis98]. Fontana has derived a simulation of the Moog filter, which avoids the extra delay in the feedback loop [Fon07]. Instead, Fontana directly solves the contribution of the states of first-order lowpass filters to the output signal. This algorithm requires more operations than the version that was described above, because divisions are involved. However, it is likely to be a more faithful simulation of the Moog ladder filter, since the structure is in principle identical to the analog prototype system.

Other researchers have considered the nonlinear behavior of the Moog filter. It is well known that the transistors used in analog filters softly saturate the signal, when the signal level becomes

high. Huovilainen [Huo04] first derived a physically correct nonlinear model for the Moog filter, including the nonlinear behavior of all transistors involved. The model contains five instantaneous nonlinear functions in a feedback loop, one for each first-order section and one for the feedback. It was then found that oversampling by a factor of two at least is necessary to reduce aliasing when nonlinear functions are used [Huo04]. The structure self-oscillates when the feedback gain is 1.0. The feedback gain can even be larger than that; the nonlinear functions prevent the system from becoming unstable. The nonlinear functions also provide amplitude compression, because large signal values are limited by each of them.

Figure 12.4 shows a simplified version of Huovilainen's nonlinear Moog filter model, in which only one nonlinear function is used [VH06]. The ideal form of the nonlinear function in this case is the hyperbolic tangent function [Huo04]. In real-time implementations this would usually be implemented using a look-up table. Alternatively, another similar smoothly saturating function, such as a third-order polynomial, can be used instead. A **MATLAB** implementation of the simplified nonlinear Moog filter is given in M-file 12.2.

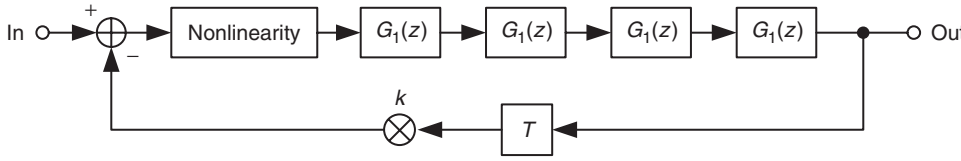

Figure 12.4 A simplified nonlinear Moog filter structure proposed in [VH06].

The simplified nonlinear Moog filter may sound slightly different from the full model. However, they share many common features, such as the ability to self-oscillate (when the resonance parameter is set to a value larger than 1) and similar compression characteristics. The latter feature is demonstrated in Figure 12.5, in which a so-called overtone sweep has been generated by filtering a sawtooth signal using three different digital versions of the Moog filter (the signals were published in the *Computer Music Journal DVD* in 2006 as demonstrations related to [VH06]). The resonant frequency of the filters slowly decreases with time. In Figure 12.5(a), the signal level increases every time the resonance frequency coincides with one of the harmonics of the signal. However, the nonlinear compression decreases these amplitude variations in Figures 12.5(b) and (c). There are minor differences between the two signals, but the output signal envelopes of the two nonlinear models are fairly similar.

M-file 12.2 (moogvcf.m)

```
function [out,out2,in2,g,h0,h1] = moogvcf(in,fc,res)
% function [out,out2,in2,g,h0,h1] = moogvcf(in,fc,res)
% Authors: Välimäki, Bilbao, Smith, Abel, Pakarinen, Berners
% Parameters:
% in = input signal
% fc= cutoff frequency (Hz)
% res = resonance (0...1 or larger for self-oscillation)
fs = 44100;  % Input and output sampling rate
fs2 = 2*fs;  % Internal sampling rate
% Two times oversampled input signal:
in = in(:); in2 = zeros(1,2*length(in)); in2(1:2:end) = in;
h = fir1(10,0.5); in2 = filter(h,1,in2);  % Anti-imaging filter
Gcomp = 0.5;  % Compensation of passband gain
g = 2*pi*fc/fs2;  % IIR feedback coefficient at fs2
Gres = res;  % Direct mapping (no table or polynomial)
```

```
h0 = g/1.3; h1 = g*0.3/1.3; % FIR part with gain g
w = [0 0 0 0 0]; % Five state variables
wold = [0 0 0 0 0];  % Previous states (unit delays)
out = zeros(size(in)); out2 = zeros(size(in2));
for n = 1:length(in2),
    u = in2(n) - 4*Gres*(wold(5) - Gcomp*in2(n));  % Input and feedback
    w(1) = tanh(u);  % Saturating nonlinearity
    w(2) = h0*w(1) + h1*wold(1) + (1-g)*wold(2);  % First IIR1
    w(3) = h0*w(2) + h1*wold(2) + (1-g)*wold(3);  % Second IIR1
    w(4) = h0*w(3) + h1*wold(3) + (1-g)*wold(4);  % Third IIR1
    w(5) = h0*w(4) + h1*wold(4) + (1-g)*wold(5);  % Fourth IIR1
    out2(n) = w(5);  % Filter output
    wold = w;  % Data move (unit delays)
end
out2 = filter(h,1,out2);  % Antialiasing filter at fs2
out = out2(1:2:end);  % Decimation by factor 2 (return to original fs)
```

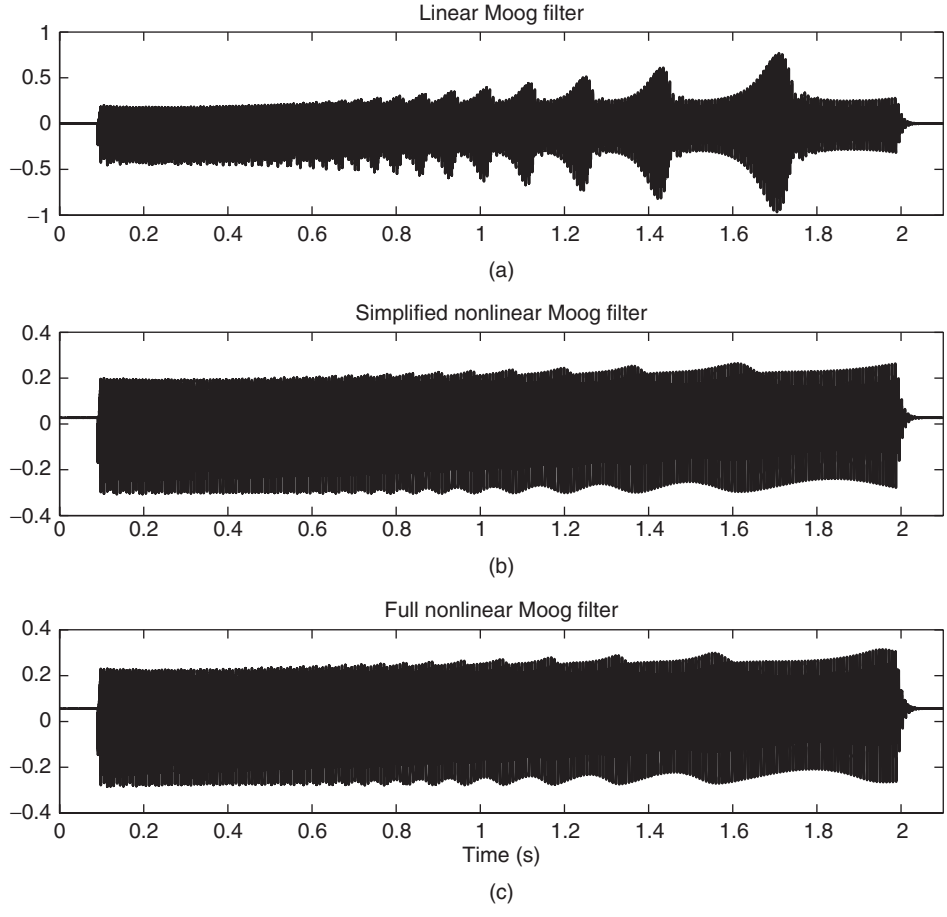

Figure 12.5 Sawtooth signal filtered with three Moog filter models: (a) the linear Stilson – Smith model of Figure 12.3, (b) the simplified nonlinear model of Figure 12.4, and (c) the full nonlinear model [Huo04].

12.2.3 Tone stack

As another useful example of a virtual analog filter, a three-channel equalizing filter used in electric guitar and bass amplifiers is presented. This so-called tone stack has the following linear third-order transfer function,

$$H_{\text{tone}}(z) = \frac{B_0 + B_1 z^{-1} + B_2 z^{-2} + B_3 z^{-3}}{A_0 + A_1 z^{-1} + A_2 z^{-2} + A_3 z^{-3}}. \tag{12.3}$$

Yeh has derived the parameter values for this digital tone stack model [YS06, Yeh09]. The equations can be seen in the M-file 12.3 below. Three parameters (top, mid, and low), which all affect several filter coefficients, can be varied to adjust the filter's magnitude response.

Figure 12.6 shows examples of magnitude responses with different parameter settings. Figure 12.6(a), (b), and (c) present the variations in the low-, middle-, and high-frequency regions, respectively. Note that the parameters are not strictly independent, but they all affect the filter gain in the middle range (see, for example, the magnitude response around 1 kHz). Furthermore, the neutral setting (low = mid = top = 0.5) does not yield a flat response, see dashed lines in Figure 12.6. This filter has its own character.

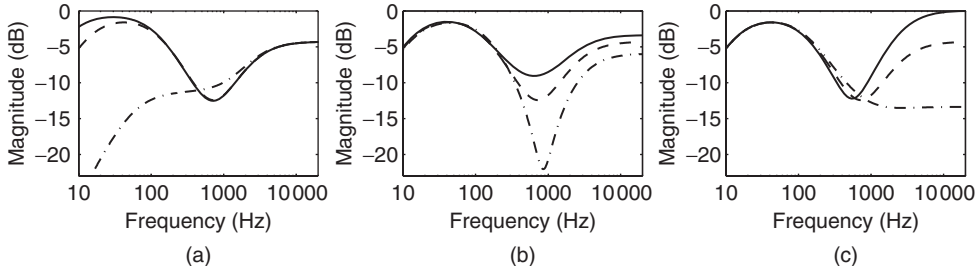

Figure 12.6 Magnitude responses of the tone stack filter model with the following parameters: (a) low = 0 (dash-dot line), low = 0.5 (dashed line), low = 1 (solid line), (b) mid = 0 (dash-dot line), mid = 0.5 (dashed line), mid = 1 (solid line), (c) top = 0 (dash-dot line), top = 0.5 (dashed line), top = 1 (solid line). The rest of the parameters are set to 0.5 in each case.

M-file 12.3 (bassman.m)

```
function [y,B0,B1,B2,B3,A0,A1,A2,A3] = bassman(low, mid, top, x)
% function [y,B0,B1,B2,B3,A0,A1,A2,A3] = bassman(low, mid, top, x)
% Authors: Välimäki, Bilbao, Smith, Abel, Pakarinen, Berners
%
% Parameters:
% low = bass level
% mid = noise level
% top = treble level
% x = input signal
fs = 44100;  % Sample rate
C1 = 0.25*10^-9;C2 = 20*10^-9;C3 = 20*10^-9;  % Component values
R1 = 250000;R2 = 1000000;R3 = 25000;R4 = 56000;  % Component values
% Analog transfer function coefficients:
b1 = top*C1*R1 + mid*C3*R3 + low*(C1*R2 + C2*R2) + (C1*R3 + C2*R3);
b2 = top*(C1*C2*R1*R4 + C1*C3*R1*R4) - mid^2*(C1*C3*R3^2 + C2*C3*R3^2) ...
    + mid*(C1*C3*R1*R3 + C1*C3*R3^2 + C2*C3*R3^2) ...
    + low*(C1*C2*R1*R2 + C1*C2*R2*R4 + C1*C3*R2*R4) ...
```

```
    + low*mid*(C1*C3*R2*R3 + C2*C3*R2*R3) ...
    + (C1*C2*R1*R3 + C1*C2*R3*R4 + C1*C3*R3*R4);
b3 = low*mid*(C1*C2*C3*R1*R2*R3 + C1*C2*C3*R2*R3*R4) ...
    - mid^2*(C1*C2*C3*R1*R3^2 + C1*C2*C3*R3^2*R4) ...
    + mid*(C1*C2*C3*R1*R3^2 + C1*C2*C3*R3^2*R4) ...
    + top*C1*C2*C3*R1*R3*R4 - top*mid*C1*C2*C3*R1*R3*R4 ...
    + top*low*C1*C2*C3*R1*R2*R4;
a0 = 1;
a1 = (C1*R1 + C1*R3 + C2*R3 + C2*R4 + C3*R4) + mid*C3*R3 ...
    + low*(C1*R2 + C2*R2);
a2 = mid*(C1*C3*R1*R3 - C2*C3*R3*R4 + C1*C3*R3^2 + C2*C3*R3^2) ...
    + low*mid*(C1*C3*R2*R3 + C2*C3*R2*R3) ...
    - mid^2*(C1*C3*R3^2 + C2*C3*R3^2) ...
    + low*(C1*C2*R2*R4 + C1*C2*R1*R2 + C1*C3*R2*R4 + C2*C3*R2*R4) ...
    + (C1*C2*R1*R4 + C1*C3*R1*R4 + C1*C2*R3*R4 + C1*C2*R1*R3 ...
    + C1*C3*R3*R4 + C2*C3*R3*R4);
a3 = low*mid*(C1*C2*C3*R1*R2*R3 + C1*C2*C3*R2*R3*R4) ...
    - mid^2*(C1*C2*C3*R1*R3^2 + C1*C2*C3*R3^2*R4) ...
    + mid*(C1*C2*C3*R3^2*R4 + C1*C2*C3*R1*R3^2 ...
    - C1*C2*C3*R1*R3*R4) + low*C1*C2*C3*R1*R2*R4 + C1*C2*C3*R1*R3*R4;
% Digital filter coefficients:
c = 2*fs;
B0 = -b1*c - b2*c^2 - b3*c^3; B1 = -b1*c + b2*c^2 + 3*b3*c^3;
B2 = b1*c + b2*c^2 - 3*b3*c^3; B3 = b1*c - b2*c^2 + b3*c^3;
A0 = -a0 - a1*c - a2*c^2 - a3*c^3;
A1 = -3*a0 - a1*c + a2*c^2 + 3*a3*c^3;
A2 = -3*a0 + a1*c + a2*c^2 - 3*a3*c^3;
A3 = -a0 + a1*c - a2*c^2 + a3*c^3;
y = filter(B,A,x);  % Output signal
```

12.2.4 Wah-wah filter

The wah-wah filter was introduced in Section 2.4.1. It operates by sweeping a single resonance through the spectrum. Typically this resonance is second-order. Following [Smi08], this section will describe digitization of the "CryBaby" wah-wah filter controlled by footpedal.

Figure 12.7(a), (b), and (c) show the amplitude responses (solid lines) of a CryBaby wah pedal measured at three representative pedal settings (rocked fully backward, middle, and forward). Our goal is to "digitize" the CryBaby by devising a sweeping resonator that audibly matches these three responses when the "wah" variable is 0, 1/2, and 1, respectively. The results of this exercise are shown in dashed lines in 12.7(a)–(c), and the audible quality is excellent.

Based on the measured shape of the amplitude response (a bandpass-resonator characteristic), and knowledge (from circuit schematics) that the bandpass is second-order, the transfer function can be presumed to be of the form

$$H(s) = g\frac{s-\xi}{\left(\frac{s}{\omega_r}\right)^2 + \frac{2}{Q}\left(\frac{s}{\omega_r}\right) + 1}, \tag{12.4}$$

where g is an overall gain factor, ξ is a real zero at or near dc (the other being at infinity), ω_r is the pole resonance frequency, and Q is the so-called "quality factor" of the resonator [Smi07].[1] The measurements reveal that ω_r, Q, and g all vary significantly with pedal angle θ. As discussed in [Smi08], good choices for these functions are as shown in M-file 12.4, where the controlling `wah` variable is the pedal-angle θ normalized to a [0, 1] range.

[1] https://ccrma.stanford.edu/~jos/filters/Quality_Factor_Q.html

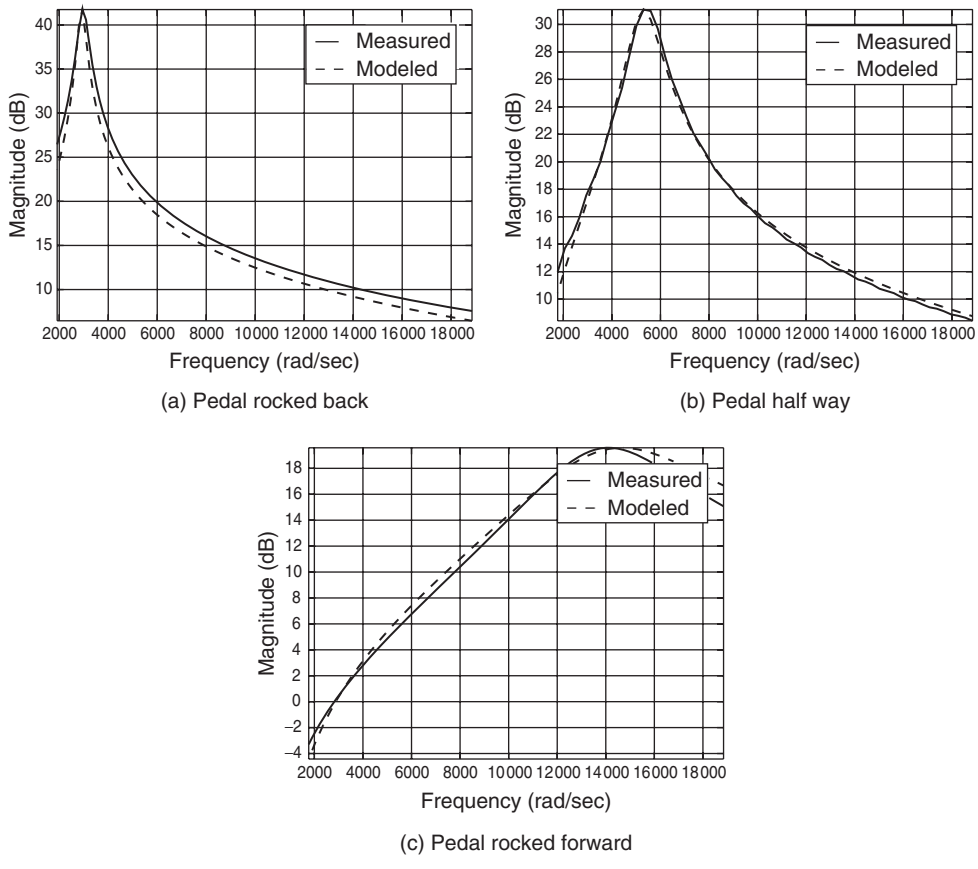

Figure 12.7 Measured (solid) and modeled (dashed) amplitude responses of the CryBaby wah pedal at three different pedal angles [Smi08].

M-file 12.4 (wahcontrols.m)

```
function [g,fr,Q] = wahcontrols(wah)
% Authors: Välimäki, Bilbao, Smith, Abel, Pakarinen, Berners
% function [g,fr,Q] = wahcontrols(wah)
%
% Parameter: wah = wah-pedal-angle normalized to lie between 0 and 1
g  = 0.1*4^wah;       % overall gain for second-order s-plane transfer funct.
fr = 450*2^(2.3*wah); % resonance frequency (Hz) for the same transfer funct.
Q  = 2^(2*(1-wah)+1); % resonance quality factor for the same transfer funct.
```

Digitization

Closed-form expressions for digital filter coefficients in terms of (Q,fr,g) based on $z = e^{sT} \approx 1 + sT$ (low-frequency resonance assumed) yield the code shown in M-file 12.5.

M-file 12.5 (wahdig.m)

```
% wahdig.m
% Authors: Välimäki, Bilbao, Smith, Abel, Pakarinen, Berners
```

```
% A = wahdig(fr,Q,fs)
%
% Parameters:
% fr = resonance frequency (Hz)
% Q  = resonance quality factor
% fs = sampling frequency (Hz)
%
% Returned:
% A = [1 a1 a2] = digital filter transfer-function denominator poly

frn = fr/fs;
R = 1 - PI*frn/Q; % pole radius
theta = 2*PI*frn; % pole angle
a1 = -2.0*R*cos(theta); % biquad coeff
a2 = R*R;               % biquad coeff
```

Note that in practice each time-varying coefficient should be smoothed, e.g., by a unity-gain one-pole smoother with pole at $p = 0.999$:

$$H_s(z) = \frac{1-p}{1-p\,z^{-1}}. \tag{12.5}$$

Virtual CryBaby results

While the presented wah pedal model sounds very faithful to the original (minus its noise), at low resonance frequencies the loudness is significantly greater than at high resonance frequencies. (This is a characteristic of the original wah pedal.) Therefore, an improvement over the original could be to determine a new scaling function `g(wah)` that preserves *constant loudness* as much as possible as the pedal varies. A similar effect can be had by applying dynamic range compression to the wah output, as is often done when recording an electric guitar.

A Faust[2] software implementation of the CryBaby wah pedal described in this section is included in the Faust distribution (file `effect.lib`).

12.2.5 Phaser

The phasing effect was introduced in Section 2.4.2. It operates by sweeping a few "notches" through the input signal spectrum. The notches are created by summing the input signal with a variably phase-shifted version of the input signal, as shown in Figure 12.8. The phase-shifting stages are conventionally first- or second-order *allpass filters* [Smi07].[3]

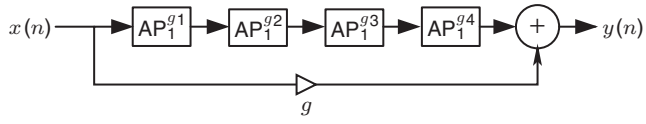

Figure 12.8 Phaser implemented by summing the direct signal with the output of four first-order allpass filters in series (from [Smi10]).

[2] `http://faust.grame.fr/`

[3] `https://ccrma.stanford.edu/~jos/filters/Analog_Allpass_Filters.html`

In analog hardware, such as the Univibe or MXR phase shifters, the allpass filters are typically first-order. Thus, each analog allpass has a transfer function of the form

$$H_a(s) = -\frac{s - \omega_b}{s + \omega_b}, \tag{12.6}$$

where we will call ω_b (a real number) the *break frequency* of the allpass.

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

$$\mathrm{AP}_1^{g_i}(z) \triangleq -\frac{g_i + z^{-1}}{1 + g_i z^{-1}}. \tag{12.7}$$

Thus, we wish to "digitize" each first-order allpass by means of a mapping from the s plane to the z plane. There are several ways to accomplish this goal [RG75]. In this case, an excellent choice is the *bilinear transformation*[4] defined by

$$s \;\rightarrow\; c\frac{z-1}{z+1}, \tag{12.8}$$

where c is chosen to map one particular analog frequency to a particular digital frequency (other than dc or half the sampling rate, which are always mapped from dc and infinity in the s plane, respectively). In this case, c is well chosen for each section to map the *break frequency* of the section to the corresponding point on the digital frequency axis. The relation between analog frequency ω_a and digital frequency ω_d follows immediately from Equation (12.8):

$$j\omega_a = c\frac{e^{jw_dT} - 1}{e^{jw_dT} + 1} = jc\frac{\sin(\omega_d T/2)}{\cos(\omega_d T/2)} = jc\tan(\omega_d T/2). \tag{12.9}$$

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

$$c = \omega_b \cot\left(\frac{\omega_b T}{2}\right). \tag{12.10}$$

The bilinear transform preserves filter order (so we will obtain a first-order digital allpass for each first-order analog allpass), and it always maps a stable analog filter to a stable digital filter. In fact, the entire $j\omega$ axis of the s plane maps to the unit circle of the z plane, giving a one-to-one correspondence between the analog and digital frequency axes. The main error in the bilinear transform is its frequency warping, which is displayed in Equation (12.9). Only dc and one other finite frequency (chosen by setting c) are mapped without any warping error.

Applying the bilinear transformation Equation (12.8) to the first-order analog allpass filter Equation (12.6) gives

$$H_d(z) = H_a\left(c\frac{1-z^{-1}}{1+z^{-1}}\right) = \frac{c\left(\frac{1-z^{-1}}{1+z^{-1}}\right) - \omega_b}{c\left(\frac{1-z^{-1}}{1+z^{-1}}\right) + \omega_b} \triangleq \frac{p_d - z^{-1}}{1 - p_d z^{-1}}, \tag{12.11}$$

where we have denoted the pole of the digital allpass by

$$p_d \triangleq \frac{c - \omega_b}{c + \omega_b} = \frac{1 - \tan(\omega_b T/2)}{1 + \tan(\omega_b T/2)} \;\approx\; \frac{1 - \omega_b T/2}{1 + \omega_b T/2} \;\approx\; 1 - \omega_b T. \tag{12.12}$$

[4] https://ccrma.stanford.edu/~jos/pasp/Bilinear_Transformation.html

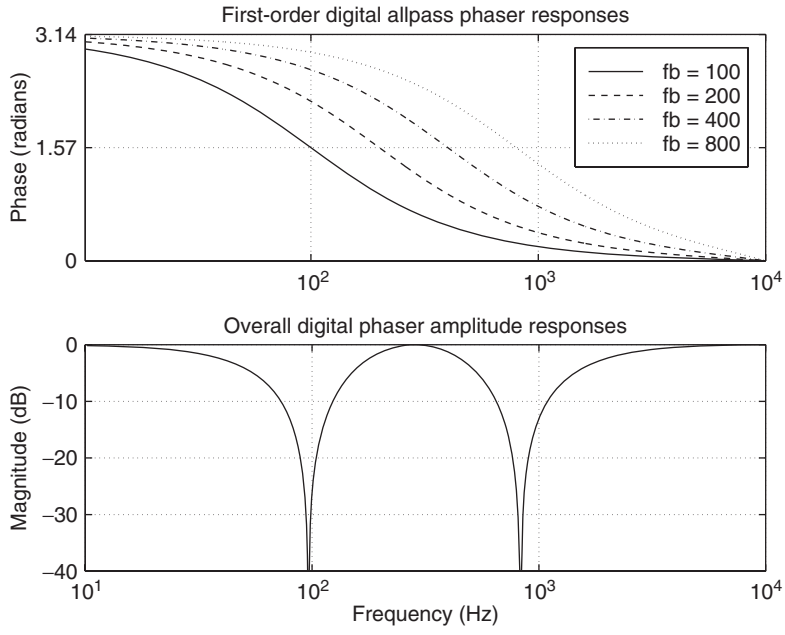

Figure 12.9 (a) Phase responses of first-order digital allpass sections having break frequencies at 100, 200, 400, and 800 Hz. The sampling rate is 20 kHz. (b) Corresponding phaser amplitude response [Smi10].

Figure 12.9 shows the digital phaser response curves. They look almost identical to the analog response curves. 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 Figure 12.10. We can see that the phase responses of the analog and digital allpass chains diverge visibly only above 9 kHz. The analog phase response approaches zero in the limit as $\omega_a \to \infty$, 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 to digitize an analog system.

In general, the bilinear transform works well to digitize feedforward analog structures in which the high-frequency warping is acceptable. When frequency warping is excessive, it can be alleviated by the use of *oversampling*; for example, the slight visible deviation in Figure 12.10 below 10 kHz can be largely eliminated by increasing the sampling rate by 15% or so. See 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 under the bilinear transform [SS96, Sti06]. In such cases it is common to insert an extra unit delay in the loop, which has little effect at low frequencies. See Section 12.2.2. (Stability should then be carefully checked, as it is no longer guaranteed.)

Phasing with second-order allpass filters

While the use of first-order allpass sections is classic for hardware phase shifters, second-order allpass filters, mentioned in Section 2.4.2, are easier to use for generating precisely located notches that are more independently controllable [Smi84].

The architecture of a phaser based on second-order allpasses is identical to that in Figure 12.8, but with each first-order allpass $\mathrm{AP}_1^{g_i}$ being replaced by a second-order allpass $\mathrm{AP}_2^{R_i,\theta_i}$, where the control parameters R_i and θ_i are given below. As before, the phaser will have a notch wherever

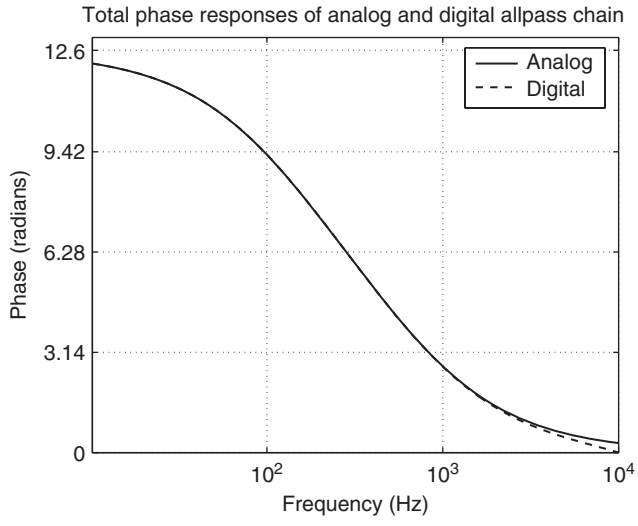

Figure 12.10 Phase response of four first-order allpass sections in series – analog and digital cases overlaid [Smi10].

the phase of the allpass chain passes through π radians (180 degrees). It can be shown that for second-order allpasses notch frequencies occur close to the resonant frequencies of the allpass sections [Smi84]. It is therefore convenient to use allpass sections of the form

$$H(z) = \frac{a_2 + a_1 z^{-1} + z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}, \tag{12.13}$$

where

$$a_1 = -2R\cos(\theta),$$
$$a_2 = R^2,$$

$R < 1$ is the radius of each pole in a complex-conjugate pole pair, and the pole angles are $\pm\theta$. The pole angle $\theta \in (0, \pi)$ can be interpreted as $\theta = \omega_n T$, where ω_n is the desired notch frequency and T is the sampling interval. The pole radius R controls the width of the notch – the closer it is to 1, the narrower the notch (and the more accurate is the tuning parameter θ).

12.3 Circuit-based valve emulation

Digital emulation of valve- or vacuum-tube amplifiers is currently a vibrant area of research with many commercial applications. As explained in a recent review article [PY09], most existing digital valve-emulation methods may roughly be divided into static waveshapers, custom nonlinear filters, and circuit-simulation-based techniques. The first type of these methods, static waveshapers (e.g., [Sul90, Kra91, AS96, Shi96, DMRS98, FC01, MGZ02, Jac03, Ame03, SST07, SM07]), use memoryless nonlinear functions for creating signal distortion and linear filtering before and after the nonlinearity for tuning the magnitude response. Oversampling is usually used to avoid signal aliasing.

12.3.1 Dynamic nonlinearities and impedance coupling

Although the valve component itself is mainly a memoryless device that can in principle be approximated with static nonlinear functions, reactive components (typically capacitors) in the

circuit make the nonlinearity act as dynamic. This means that in reality, the shape of the nonlinearity changes according to the input signal and the internal state of the circuit. Custom nonlinear valve emulation filters [Pri91, KI98, GCO+04, KMKH06] simulate this dynamic nonlinearity, for example by creating a feedback loop around the nonlinearity.

Another important phenomenon in real valve circuits is the two-directional impedance coupling between components and different parts of the circuit. This causes, for example, an additional signal-dependent bias variation of a valve circuit when connected to a reactive load, such as a loudspeaker. If the digital valve circuit model uses a unidirectional signal path – as many simple models do – altering some part in the virtual circuit has no effect on the parts earlier in the signal chain. For example, if a linear loudspeaker model is attached to a virtual tube circuit with unidirectional signal flow, the resulting effect will only be a linear filtering according to the transfer characteristics of the loudspeaker, and no coupling effects with the tube circuit will be present. Nonlinear circuit-simulation-based modeling techniques (e.g., [Huo04, Huo05, YS08, YAAS08, Gal08, MS09]) try to incorporate the impedance coupling effect, at least between some parts of the circuit. Traditionally, these methods use Kirchhoff's rules and energy conservation laws to manually obtain the ordinary differential equations (ODEs) that represent the operation of the circuit. The ODEs are then discretized (usually using the bilinear transform), and the system of implicit nonlinear equations is iteratively solved using numerical integration methods, such as the Newton – Raphson or Runge – Kutta.

12.3.2 Modularity

From the digital valve-amplifier designer's point of view, modularity would be a desirable property for the emulator. In an ideal system, it should be easy to edit the circuit topology, for example by graphically altering the circuit schematics, and the sonic results should be immediately available. Enabling full control over the digital circuit construction would allow the emulation of any vintage valve amplifier simply by copying its circuit schematics into the system. Furthermore, it would enable the designer to apply the knowledge of valve-amplifier building tradition into the novel digital models. Note that this would mean that the designer should not be limited by conventional circuit design constraints or even general physical laws in creating a new digital amplifier, so that also purely digital or "abstract" processing techniques could well be used in conjunction. In the late 1990s, several valve circuit models (for example [Ryd95, Lea95, PLBC97, Sju97]) were presented for the SPICE circuit simulator software [QPN+07]. Although these models were modular and able to simulate the full impedance coupling within the whole system, they could not be used as real-time effects due to their intensive computational load. In fact, even over a decade later, these SPICE models are computationally still too heavy to run in real time.

Wave digital filters (WDFs), introduced by Fettweis [Fet86], offer a modular and relatively light approach for real-time circuit simulation. The main difference between WDFs and most other modeling methods is that WDFs represent all signals and states as wave variables and use local discretization for the circuit components. The WDF method will be explained more thoroughly in Section 12.3.3, and a simple nonlinear circuit simulation example using WDFs is presented in Section 12.3.4. The K-method, a state-space approach for the simulation of nonlinear systems has been introduced by Borin and others [BPR00]. This alternative approach for circuit simulation is currently another promising candidate for real-time valve simulation. It should be noted that although state-space models are not inherently modular, a novel technique by Yeh [Yeh09] allows the automatization of the model-building process, so that state-space simulation models may automatically be obtained from a circuit schematics description.

12.3.3 Wave digital filter basics

The basics of WDF modeling are briefly reviewed in the following section. A signal-processing approach has been chosen with less emphasis on the physical modeling aspects, in order to clarify

the operation of the modeling example in Section 12.3.4. For a more thorough tutorial on WDFs, see, for example [Fet86, VPEK06].

One-port elements

WDF components connect to each other using ports. Each port has two terminals, one for transporting incoming waves, and one for transporting outgoing waves. Also, each port has an additional parameter, port resistance, which is used in implementing proper impedance coupling between components. The relationship between the Kirchhoff pair (e.g., voltage U and current I) and wave variables A and B is given by

$$\begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} 1 + R_\mathrm{p} \\ 1 - R_\mathrm{p} \end{bmatrix} \begin{bmatrix} U \\ I \end{bmatrix} \Leftrightarrow \begin{bmatrix} U \\ I \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1/R_\mathrm{p} & -1/R_\mathrm{p} \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix}, \tag{12.14}$$

where R_p denotes the port resistance.

Note that this port resistance is purely a computational parameter, and it should not be confused with electrical resistance. Most elementary circuit components can be represented using WDF one-port elements. Some basic circuit components are illustrated together with their WDF one-port counterparts in Figure 12.11. Some other circuit components, such as transformers, cannot be represented as one-port elements, but require a multiport representation, which is out of the scope of this book. The port resistances of the one-port elements in Figure 12.11 can be given as follows:

$$R_\mathrm{p} = \begin{cases} R & \text{for resistance,} \\ 1/(2CF_\mathrm{s}) & \text{for capacitance,} \\ 2LF_\mathrm{s} & \text{for inductance,} \end{cases} \tag{12.15}$$

where R, C and L are the electrical resistance (Ohms), capacitance (Farads), and inductance (Henrys), respectively, while F_s stands for the sample rate (Hertz). Similar to the WDF resistor, the port resistance of a voltage source is equivalent to the physical resistance.

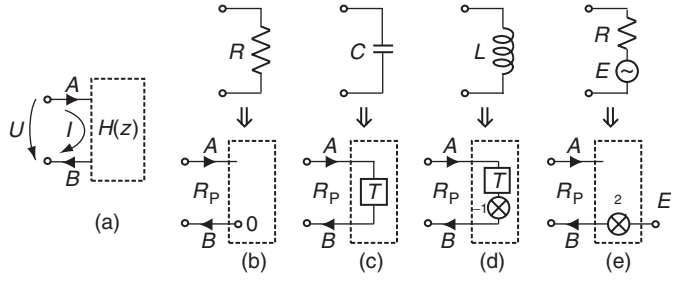

Figure 12.11 Basic WDF one-port elements: (a) a generic one-port with voltage U across and current I across the terminals, (b) resistor, (c) capacitor, (d) inductor, (e) voltage source. Here, A represents an incoming wave for each element, while B denotes an outgoing wave. Symbol R_p stands for the port resistance. Figure adapted from [PK10] and reprinted with IEEE permission.

Adaptors

The WDF circuit components connect to each other via adaptors. In practice, the adaptors direct signal routing inside the WDF circuit model, and implement the correct impedance coupling via port resistances. Although the number of ports in an adaptor is not limited in principle, three-port adaptors are typically used, since any N-port adaptor ($N > 3$) can be expressed using a connection of three-port adaptors, so that the whole WDF circuit becomes a binary tree structure [DST03].

There are two types of WDF adaptors: series and parallel, which implement the series and parallel connection between elements, respectively. Furthermore, the port resistance values for the adaptors should be set equal to the port resistances of the elements they are connected to.

Figures 12.12(a) and (b) illustrate the WDF series and parallel adaptors, respectively. Here, the ports have been numbered, for clarity. The outgoing wave B_n at port $n = 1, 2, 3$ of a three-port adaptor can generally be expressed as

$$B_n = \begin{cases} A_n - 2R_n(A_1 + A_2 + A_3)/(R_1 + R_2 + R_3) & \text{for series adaptor,} \\ 2(G_1A_1 + G_2A_2 + G_3A_3)/(G_1 + G_2 + G_3) - A_n & \text{for parallel adaptor,} \end{cases} \tag{12.16}$$

where A_n denotes the incoming wave at port n and $G_n = 1/R_n$ is the inverse of the port resistance R_n.

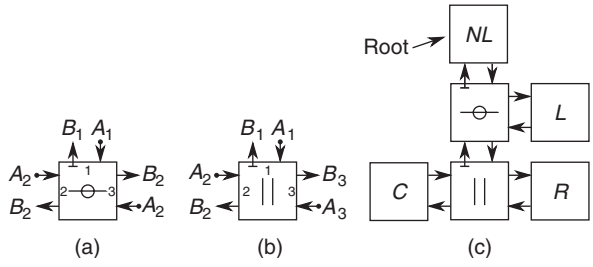

Figure 12.12 WDF three-port serial (a) and parallel (b) adaptors and an example WDF binary tree (c).

Computational scheduling

Figure 12.12(c) depicts an example WDF binary tree, simulating an RLC circuit with a nonlinear resistor (marked NL in the figure). As can be seen, the adaptors act as nodal points connecting the one-port elements together. For this reason, the adaptors in a WDF binary tree are also called nodes, and the one-port elements are called the leaves of the binary tree. When deciding the order of computations, a single one-port element must be chosen as the root of the tree. In Figure 12.12(c), the nonlinear resistor is chosen as the root. When this decision has been made, the WDF simulation starts by first evaluating all the waves propagating from the leaves towards the root. When the incoming wave arrives at the root element, the outgoing wave is computed, and the waves are propagated from the root towards the leaves, after which the process is repeated.

As can be seen in Equation (12.16), the wave leaving the WDF adaptor is given as a function of the incoming waves and port resistances. This poses a problem for the computation schedule discussed above, since in order for the adaptor to compute the waves towards the root, it would also have to know the wave coming from the root at the same time. Interestingly, the port resistances can be used as a remedy. By properly selecting the port resistance for the port facing the root, the wave traveling towards the root can be made independent from the wave traveling away from the root.

For example, if we name the port facing the root element as port number one, and set its port resistance as $R_1 = R_2 + R_3$ if it is a series adaptor (and the inverse port resistance $G_1 = G_2 + G_3$ if it is a parallel adaptor), Equation (12.16) simplifies into

$$B_1 = \begin{cases} -A_2 - A_3 & \text{for series connection} \\ G_2/(G_2 + G_3)A_2 + G_3/(G_2 + G_3)A_3 & \text{for parallel connection} \end{cases} \tag{12.17}$$

for the wave components traveling towards the root. In our example, the port number one would be called adapted, or reflection free, since the outgoing wave does not depend on the incoming

(reflected) wave. Such adapted ports are typically denoted with a "T-shaped" ending for the outwards terminal, as illustrated for port number one in Figure 12.12(a) and (b). As Figure 12.12(c) shows, all adapted ports point towards the root.

Since the root element is connected to an adapted port, and the port resistances between the adapted port and the root should be equal, an interesting paradox arises. On one hand, the shared port resistance value should be set as required by the adaptor (for example as the sum of the other port resistances on a series adaptor), but on the other hand, the port resistance should be set as dictated by the root element (for example as a resistance value in Ohms for a resistor). If a resistor is chosen as a root element, the solution is to define the outgoing wave B from the root as [Kar]

$$B = \frac{R - R_\mathrm{p}}{R + R_\mathrm{p}} A, \tag{12.18}$$

where R is the electrical resistance of the root, R_p is the port resistance set by the adapted port, and A is the wave going into the root element. Since the port resistance of the adapted port is independent of the electrical resistance of the root, the latter can freely be varied during simulation without encountering computability problems. Thus, if the circuit has a nonlinear one-port element, it should be chosen as the root for the WDF tree (since nonlinearity can be seen as parametric variation at a signal rate). For all other leaves, changing the port resistance during simulation is problematic since correct operation would require the port resistance changes to be propagated throughout the whole tree and iterated to correctly set the changed wave values. In practice, however, it is possible to vary the port resistances (i.e., component values) of the leaves without problems, provided that the parametric variation is slow compared to the sampling rate of the system.

Nonlinearities and model initialization

In a nonlinear resistor, the electrical resistance value depends nonlinearly on the voltage over the resistor, and a correct simulation of this implicit nonlinearity requires iterative techniques in general. Note the difference between the earlier discussed global iteration for setting the values of all port resistances, and the local iteration to set only the value of the nonlinear resistor: although the adapted ports remove the need for the former, one should still in principle perform the latter. Since run-time iteration is a computationally intensive operation, a typical practical simplification for avoiding local iteration is to insert an additional unit delay to the system, for example by using the voltage value at the previous time instant when calculating the resistance value. The error caused by this extra delay is usually negligible, provided that the shape of the nonlinearity is relatively smooth, as is the case with typical valve components. Further information on nonlinear WDFs can be found in [SD99, Bil01]. Valve simulation using WDFs was first presented in [KP06], and refined models have been discussed in [PTK09, PK10].

Reactive WDF circuit components should be correctly initialized before the simulation starts. This means that the one-sample memory of inductors and capacitors should be set so that the voltages and currents comply with Kirchhoff's laws. For reactive components that have nonzero initial energy (for example an initially charged capacitor), this can be tricky since the relationship between the stored energy and the wave value inside the memory is not straightforward. However, the correct initialization procedure can still be performed by using Thévenin or Norton equivalents. A practical alternative is to set the memory of reactive WDF components to zero, and let the simulation run for some "warm-up" time before inserting the actual input signal so that a steady operating point is reached and the reactances have been properly charged. In a sense, this is similar to letting a real electric circuit warm up and reach the correct operating point before feeding the input signal.

Remaining challenges in WDF modeling

Although WDFs have many desirable properties in circuit-simulation applications, there are also some open research problems related to them that limit their usability on a large scale. Probably the

most severe limitation is that those circuit topologies that do not map onto binary trees are difficult to realize. Such topologies include bridge connections and loop-like structures. Modeling of bridge connections using WDFs is further discussed in [Kar08]. Another limitation is that a WDF binary tree can generally have only a single nonlinear element. If multiple nonlinearities are to be simulated using WDFs, global iteration should be applied for the whole sub-tree connecting the nonlinearities to each other. Alternatively, unit delays can be inserted into the system to enable computability, as done in [PK10], but this may drastically affect the stability of the system in some cases.

12.3.4 Diode circuit model using wave digital filters

This section discusses a WDF representation of a simple valve diode circuit, depicted in Figure 12.13(a). The WDF equivalent of the electric circuit is illustrated in Figure 12.13(b). The simulator algorithm is implemented using object-oriented programming for illustrating the inherent modularity of WDFs. The related code requires a basic understanding of object-oriented programming principles, and is mainly intended to show the possibilities of WDF modeling, rather than trying to provide a computationally efficient implementation. The class files are defined in the M-file 12.6, and the diode circuit model is presented in M-file 12.7. The presented WDF simulator consists of seven classes, three of which are abstract (i.e., they only serve as superclasses to real classes). It must be noted that all the presented classes are shown in a single M-file for compactness although in practice **MATLAB** requires each class to reside in an individual file. In other words, the classes in M-file 12.6 should be split into seven different files in order for the model to run in **MATLAB**.

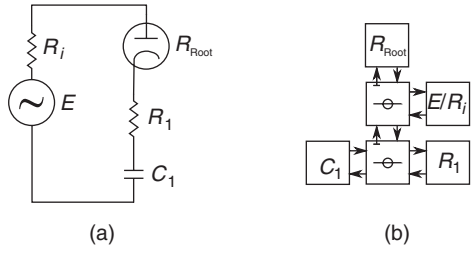

Figure 12.13 Electrical (a) and WDF (b) representations of an RC circuit with a vacuum-tube diode. The diode is denoted R_{Root}, since it is realized as a nonlinear resistor as the root element of the binary WDF tree structure.

The first class in M-file 12.6 defines the `WDF` superclass, which serves as a parent for the other six classes. The `WDF` class itself is inherited from **MATLAB**'s `hgsetget`-class, which results in all WDF elements being handle-objects. This means that when object properties (such as wave variable values) are edited, **MATLAB** only modifies the property values, instead of creating entirely new objects. The `PortRes` property defines the port resistance of a WDF object, and the `Voltage` method gives the voltage over the element, as defined in Equation (12.14). The next class in M-file 12.6 defines the `Adaptor` class, which serves as a superclass for three-port series and parallel adaptors. Two properties, `KidLeft` and `KidRight`, are defined, which serve as handles to the WDF objects connected to the adaptor.

The `ser` class defines the WDF series adaptor. The waves traveling up (towards the root) and down (away from the root) are given as properties `WU` and `WD`, respectively. The adaptor is realized so that the adapted port always points up, to enable the connection of a nonlinear element at the root. In addition to the class constructor, this class defines two methods, `WaveUp` and `set.WD`. The first of these reads the up-going wave according to Equation (12.17), and the second sets the down-going waves for the connected elements according to Equation (12.16). Note that the parallel

adaptor is similarly defined and can be found among the supplementary **MATLAB**-files on-line, although it is omitted here for brevity.

The `OnePort` class in M-file 12.6 serves as a superclass for all one-port WDF elements. It also defines the up- and down-going waves `WU` and `WD` and the `set.WD`-method. This method also updates the internal state, or memory, of a reactive WDF one-port. The last three classes in M-file 12.6 introduce the classes `R`, `C`, and `V`, which represent the WDF resistors, capacitors, and voltage sources, respectively. Note that the class `L` for implementing a WDF inductor may easily be edited from the `C` class, although it is not printed here (but included in the associated on-line code).

M-file 12.6 (`WDFClasses.m`)

```
% WDFclasses.m
% Authors: Välimäki, Bilbao, Smith, Abel, Pakarinen, Berners
%----------------------WDF Class-----------------------
classdef WDF < hgsetget % the WDF element superclass
    properties
        PortRes % the WDF port resistance
    end
    methods
            function Volts = Voltage(obj)  % the voltage (V) over a WDF element
                Volts = (obj.WU+obj.WD)/2; % as defined in the WDF literature
            end
    end;
end
%----------------------Adaptor Class-----------------------
classdef Adaptor < WDF % the superclass for ser. and par. (3-port) adaptors
    properties
        KidLeft % a handle to the WDF element connected at the left port
        KidRight % a handle to the WDF element connected at the right port
    end;
end
%----------------------Ser Class-----------------------
classdef ser < Adaptor % the class for series 3-port adaptors
    properties
        WD % this is the down-going wave at the adapted port
        WU % this is the up-going wave at the adapted port
    end;
    methods
        function obj = ser(KidLeft,KidRight) % constructor function
            obj.KidLeft = KidLeft; % connect the left 'child'
            obj.KidRight = KidRight; % connect the right 'child'
            obj.PortRes = KidLeft.PortRes+KidRight.PortRes; % adapt. port
        end;
        function WU = WaveUp(obj) % the up-going wave at the adapted port
            WU = -(WaveUp(obj.KidLeft)+WaveUp(obj.KidRight)); % wave up
            obj.WU = WU;
        end;
        function set.WD(obj,WaveFromParent) %  sets the down-going wave
            obj.WD = WaveFromParent; % set the down-going wave for the adaptor
            % set the waves to the 'children' according to the scattering rules
            set(obj.KidLeft,'WD',obj.KidLeft.WU-(obj.KidLeft.PortRes/...
            obj.PortRes)*(WaveFromParent+obj.KidLeft.WU+obj.KidRight.WU));
            set(obj.KidRight,'WD',obj.KidRight.WU-(obj.KidRight.PortRes/...
            obj.PortRes)*(WaveFromParent+obj.KidLeft.WU+obj.KidRight.WU));
        end;
    end
end
%----------------------OnePort Class-----------------------
classdef OnePort < WDF % superclass for all WDF one-port elements
    properties
        WD % the incoming wave to the one-port element
        WU % the out-going wave from the one-port element
```

```
    end;
        methods
            function obj = set.WD(obj,val) % this function sets the out-going wave
            obj.WD = val;
                if or(strcmp(class(obj),'C'),strcmp(class(obj),'L')) % if react.
                    obj.State = val; % update internal state
                end;
            end;
        end
end
%----------------------R Class-------------------------
classdef R < OnePort % a (linear) WDF resistor
        methods
            function obj = R(PortRes) % constructor function
                obj.PortRes = PortRes; % port resistance (equal to el. res.)
            end;
            function WU = WaveUp(obj) % get the up-going wave
                WU = 0; % always zero for a linear WDF resistor
                obj.WU = WU;
            end;
        end
end
%----------------------C Class-------------------------
classdef C < OnePort
    properties
        State % this is the one-sample internal memory of the WDF capacitor
    end;
        methods
            function obj = C(PortRes) % constructor function
                obj.PortRes = PortRes; % set the port resistance
                obj.State = 0; % initialization of the internal memory
            end;
            function WU = WaveUp(obj) % get the up-going wave
                WU = obj.State; % in practice, this implements the unit delay
                obj.WU = WU;
            end;
        end
end
%----------------------V Class-------------------------
classdef V < OnePort % class for the WDF voltage source (and ser. res.)
    properties
        E % this is the source voltage
    end
        methods
            function obj = V(E,PortRes) % constructor function
                obj.E = E; % set the source voltage
                obj.PortRes = PortRes; % set the port resistance
                obj.WD = 0; % initial value for the incoming wave
            end;
            function WU = WaveUp(obj) % evaluate the outgoing wave
                WU = 2*obj.E-obj.WD; % from the def. of the WDF voltage source
                obj.WU = WU;
            end;
        end
end
```

The actual diode simulation example is printed as M-file 12.7, and will be briefly explained here. M-file 12.7 starts by defining the overall simulation parameters and variables, such as the sample rate, and input and output signals. Next, the WDF elements (resistive voltage source, resistor, and a capacitor) are created, and the WDF is formed as a series connection. The nonlinear resistor is not created as an object, but it is manually implemented in the simulation loop.

The simulation loop starts by reading the input signal as the source voltage for object `V1`. Next, the `WaveUp` method is called for the WDF tree, resulting in all up-going waves to be propagated at the root. The valve resistor is modeled as the nonlinear resistance

$$R = 125.56\mathrm{e}^{-0.036U}\,\Omega \tag{12.19}$$

as a function of the voltage U over the diode. Equation (12.19) was empirically found to be a simple, but fairly accurate simulation of a GZ 34 valve diode. Here, the previous value of the diode voltage is used, in order to avoid the local iteration of the implicit nonlinearity. The wave reflection at the nonlinearity is implemented according to Equation (12.18), and the down-going waves are propagated to the leaves. Finally, the diode voltage is updated for the next time sample, and the output is read as the voltage over the resistor `R1`. After the simulation loop is finished, the results are illustrated using the plot-commands.

It is important to note, that due to the modularity of the WDFs and the object-oriented nature of the implementation, editing of the simulation circuit is very easy. All that has to be done to simulate a different circuit is to change the one-port element definitions (rows 10-13 in M-file 12.7) and the circuit topology (row 14 in M-file 12.7). In other words, M-files 12.6 and 12.7 define an entire circuit simulation software!

M-file 12.7 (`WDFDiodeExample.m`)

```
% WDFDiodeExample.m
% Authors: Välimäki, Bilbao, Smith, Abel, Pakarinen, Berners
Fs = 20000; % sample rate (Hz)
N = Fs/10; % number of samples to simulate
gain = 30; % input signal gain parameter
f0 = 100; % excitation frequency (Hz)
t = 0:N-1; % time vector for the excitation
input = gain.*sin(2*pi*f0/Fs.*t); % the excitation signal
output = zeros(1,length(input));
V1 = V(0,1); % create a source with 0 (initial) voltage and 1 Ohm ser. res.
R1 = R(80); % create an 80Ohm resistor
CapVal = 3.5e-5; % the capacitance value in Farads
C1 = C(1/(2*CapVal*Fs)); % create the capacitance
s1 =  ser(V1,ser(C1,R1)); % create WDF tree as a ser. conn. of V1,C1, and R1
Vdiode = 0; % initial value for the voltage over the diode
%% The simulation loop:
for n = 1:N % run each time sample until N
    V1.E = input(n); % read the input signal for the voltage source
    WaveUp(s1); % get the waves up to the root
    Rdiode = 125.56*exp(-0.036*Vdiode); % the nonlinear resist. of the diode
    r = (Rdiode-s1.PortRes)/(Rdiode+s1.PortRes); % update scattering coeff.
    s1.WD = r*s1.WU; % evaluate the wave leaving the diode (root element)
    Vdiode = (s1.WD+s1.WU)/2; % update the diode voltage for next time sample
    output(n) = Voltage(R1); % the output is the voltage over the resistor R1
end;
%% Plot the results
t = (1:length(input))./Fs; % create a time vector for the figure
hi = plot(t,input,'--'); hold on; % plot the input signal, keep figure open
ho = plot(t,output); hold off; % plot output signal, prevent further plots
grid on; % use the grid for clarity
xlabel('Time (s)'); ylabel('Voltage (V)'); % insert x- and y-axis labels
legend([hi ho],'Source voltage E','Voltage over R1'); % insert legend
```

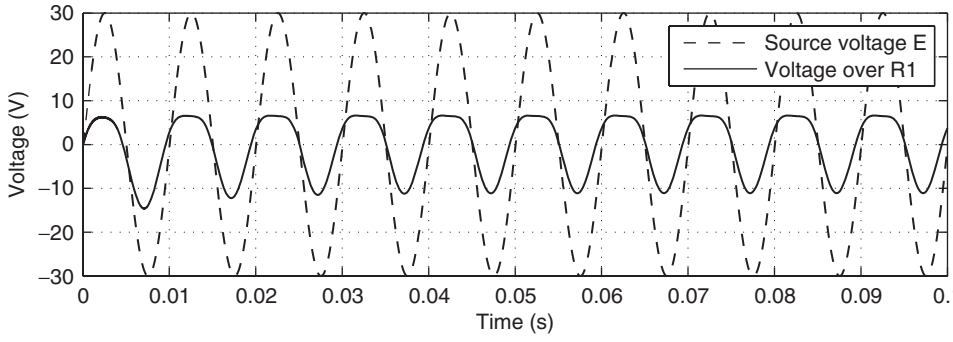

Figure 12.14 Voltage over the tube diode, as plotted by the M-file 12.7. As can be seen, the diode nonlinearity has squashed the positive signal peaks, while charging of the capacitor causes a shift in the local minima.

The output of the WDF model, i.e., the circuit response to a 100 Hz sine signal is illustrated in Figure 12.14. As expected, the positive signal peaks have been squashed since the diode lets forward current pass, reducing the potential difference over the diode. This current also charges the capacitor, reducing the voltage drop for negative signal cycles in the beginning of the simulation. Thus, Figure 12.14 visualizes both the static nonlinearity due to the diode component itself, as well as the dynamic behavior caused by the capacitance. The simulation results were verified using LTSpice simulation for the same circuit. The resulting waveforms were visually identical for both the WDF and LTSpice simulations.

12.4 Electromechanical effects

A rather special family of audio effects consists of those which employ mechanical elements in conjunction with analog electronics – though many classic effects rely on such components (such as, for example, the Leslie speaker, or tape delays), for some problems in the world of virtual analog, an involved treatment of the mechanical part of the effect is necessary, in order to be able to capture the perceptually salient quality of the effect. This is particularly the case for electromechanical artificial reverberation, which exhibits an extremely complex response. Two interesting cases, namely plate and spring reverberation, will be briefly described here.

The attribute of these devices which distinguishes them from other electromechanical effects is that the mechanical components cannot be modelled as "lumped" structurally, a spring or a plate occupies space, and its vibration pattern varies from one point to the next. It is precisely the distributed character of these components which gives these effects their complex sound – and which, at the same time, requires a somewhat different approach to emulation. One must now think of the behavior of the component in terms of various different modes of vibration, or in terms of waves which require finite propagation times, just like real acoustic spaces. Indeed, spring and plate reverbs were originally intended as convenient substitutes for real room reverbs – but developed a loyal audience of their own. Originally, such popularity was probably in spite of this distinction, but as such units have become scarce, convenient digital substitutes for these sometimes bulky and damage-prone units have become sought-after – and make for a fascinating application of virtual analog modeling!

Distributed modeling is seen only rarely in the world of digital audio effects, but plays, of course, a central role in sound synthesis based on physical models of musical instruments – a topic which is too broad to fit into this volume! In this short section, some of the basics of distributed modeling for electromechanical elements are covered, from a high-level perspective, focusing on

the nature of the phenomena involved, and some simulation techniques (and in particular, finite difference schemes). It's probably worth mentioning that distributed modeling can be a lot more expensive, in terms of number-crunching, than other analog effect emulation – which is true of any digital reverberation algorithm. But, these days, standard computer hardware is becoming powerful enough to tackle these modeling problems in real-time, or something close to it.

12.4.1 Room reverberation and the 3D wave equation

When looking at artificial reverberation techniques, it's useful to first take a look at the behaviour of real rooms – to understand the ways in which electromechanical effects attempt to emulate this real-world effect, and, more importantly, how they differ!

In air, to a very good approximation, waves travel at a constant speed, here referred to as c. c has a mild dependence on temperature, and is approximately 343 m/s at atmospheric pressure, and near room temperature [FR91]. From a given acoustic point source, waves travel uniformly in all directions (isotropically) at this speed, and there is a reduction in amplitude as the waves spread out. Formally, to a very good approximation, the time evolution of a pressure field in three dimensions may be described by the wave equation,

$$\frac{\partial^2 p}{\partial t^2} = c^2 \nabla^2 p \tag{12.20}$$

where here, $p(x, y, z, t)$ is the pressure at spatial coordinates x, y and z, and at time t. ∇^2 is the 3D Laplacian operator. For more on this equation, see [MI68].

A key concept in all distributed modeling for linear systems is a dispersion relation, relating angular frequency ω (in radians/sec) to a vector wavenumber β (measured in units of m^{-1}) for a wave component of the solution. Such a component will be of the form

$$p(x, t) = \mathrm{e}^{j\omega t + \beta \cdot x} \tag{12.21}$$

in 3D, and the dispersion relation for the wave equation above takes the following form,

$$\omega = c|\beta|. \tag{12.22}$$

It turns out to be convenient to use ω and β here, but one could equally well write this relationship in terms of frequency $f = \omega/2\pi$ and wavelength $\lambda = 2\pi/|\beta|$ – the above dispersion relation then becomes the familiar expression $c = \lambda f$. In the case of the wave equation, this expression is of a very simple form; for other systems it will not be, and often, even for a single system, there may be many such relations, which may not be expressible in simple closed form. The dispersion relation, in general, yields a huge amount of information about the behavior of a system which is very important acoustically. When a system's geometry is specified, one can say a great deal not only about acoustic attributes such as echo density and mode density, but also about how much computational power will be required to perform a digital emulation!

Expressions for wave speed as a function of frequency may be derived from a dispersion relation as

$$v_{phase} = \frac{\omega}{|\beta|} \qquad v_{group} = \frac{d\omega}{d|\beta|}, \tag{12.23}$$

where v_{phase} is the phase velocity, or propagation speed of a wave at wavenumber β, and v_{group} is the group, or packet velocity. In the case of rooms, from the dispersion relation given above, these speeds are both c. Such wave propagation is often referred to as non dispersive; all disturbances propagate at the same speed, regardless of wavelength or frequency. In the present case of wave

propagation in air, it also implies that waveforms which propagate from one point to another preserve shape, or remain coherent during propagation.

When a given delimited acoustic space is defined, the obvious acoustic effect is that of echoes. For a room with reflective walls, a typical response consists of series of distinct echoes (early reflections) which are perceptually distinct, and which serve as localization cues, followed by increasingly dense late reflections, and, finally when reflections become so dense so as to be not perceptually distinct, a reverberant tail. It is room geometry which gives acoustic responses their characteristic complexity – and this is definitely not the case for plate and spring reverberation, which emulate this response using relatively simple structures.

Digital emulation of 3D acoustic spaces through the direct solution of the wave equation is a daunting task, from the point of view of computational complexity (though it is an active research topic [Bot95, MKMS07], which will surely eventually bear fruit). In fact, by further examination of the dispersion relation one may arrive at bounds on computational cost: for any simulation of the 3D wave equation over a space of volume V, and which is intended to produce an approximation to the response over a band of frequencies from 0 Hz, to $f_s/2$ Hz, for some frequency f_s (i.e., an audio sample rate), the amount of memory required will be approximately

$$N(f_s) \propto V f_s^3/c^3, \tag{12.24}$$

where the constant of proportionality depends on the particular method employed. For $c = 340$, and at at typical audio rate $f_s = 48$ kHz, and for even a moderately sized room ($V = 1000$ m^3), this can be very large indeed – here, for (say) a finite difference scheme $N = 2.8 \times 10^9$! Because arithmetic operations will presumably be performed on all the data, locally, at the audio rate, the operation count will be on the order of $N \times f_s$ flops, which is approaching the petaflop range. For more on the topic of computational complexity estimates in physical audio applications, see [Bil09].

Most reverberation techniques are thus based around perceptual simplifications, as described in detail in Chapter 5. In industrial acoustical applications, methods based on the image source method [AB79] are popular, and rely on simplified models of reflection from surfaces; in audio, feedback delay networks [Puc82, Roc97, JC91], which are very efficient, but not based on direct solution of the 3D wave equation, are commonly employed.

12.4.2 Plates and plate reverberation

Plate reverberation was originally intended as a convenient means of applying a reverberant effect to a recorded sound [Kuh58], long before the advent of digital reverberation. Such effects were extensively researched and eventually commericialized, with the EMT-140 the most successful resulting product. The basic operation of such a unit is relatively simple: a metal plate, normally rectangular, made of steel and supported by a frame is driven by an input signal through an actuator, and outputs are read at a pair of pickups. See Figure 12.15(a). Such a device is often complemented by adjustible absorbing plates which allow for some control over the decay time. The response of a plate is generally quite different from that of room acoustic reverberation – distinct relections are notably absent, and the response as a whole has a smooth noise-like character.

A given plate is characterized by its density ρ, in kg/m^3, Young's modulus E, in kg/s^{2}m, Poisson's ratio ν (dimensionless), and geometrical properties such as its thickness H in m, and its surface area A in m^2, as well as the particular boundary conditions. A typical reverberation unit is constructed from steel, with a thickness of approximately 0.5 mm, and with a surface area of approximately 2 m^2 (other materials and dimensions are also in use – the much smaller EMT 240 unit employs thin gold foil instead of steel). Usually the edges are very nearly free to vibrate, except for at the plate supports. For a sufficiently thin lossless plate, the equation of motion may

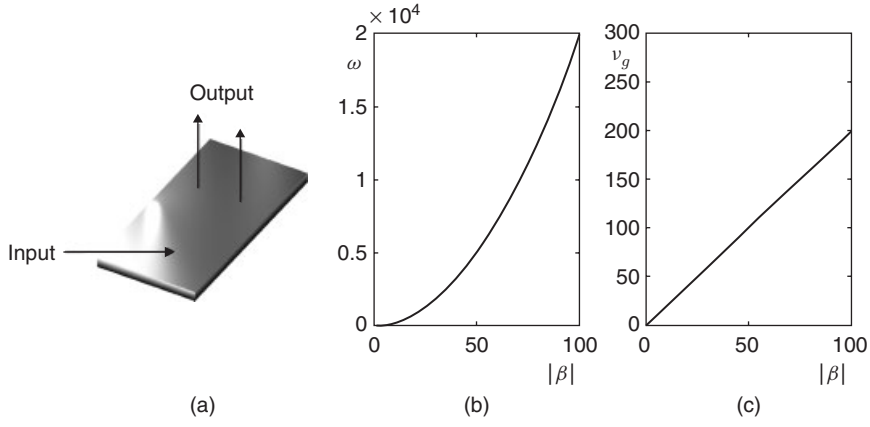

Figure 12.15 (a) A metal plate, driven by an input signal, and from which output(s) are read at distinct locations. (b) Dispersion relation, and (c) group velocity for a thin plate.

be written succinctly as

$$\frac{\partial^2 u}{\partial t^2} = -\kappa^2 \nabla^2 \nabla^2 u, \tag{12.25}$$

where ∇^2 is the 2D Laplacian operator defined, in Cartesian coordinates (x, y), as

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}. \tag{12.26}$$

For more on the physics of plates, see, e.g., [FR91, Gra75, MI68].

The dispersion relation for a plate is very different from that corresponding to wave propagation in air,

$$\omega = \kappa \beta^2, \tag{12.27}$$

where $\kappa = \sqrt{EH^2/(12(1-\nu^2)\rho}$. Now, the phase and group velocities are no longer constant,

$$v_{phase} = \kappa\beta = \sqrt{\kappa\omega} \qquad v_{group} = 2\kappa\beta = 2\sqrt{\kappa\omega} \tag{12.28}$$

See Figure 12.15 (b) and (c). As a result, wave propagation is dispersive, and highly so. See Figure 12.16, showing the time evolution of plate displacement in response to a pulse. Notice in particular that the high-frequency components of the pulse travel more quickly than the low-frequency components – and thus the pulse rapidly loses coherence as it travels. Another related distinction between plate and room responses is the mode density, which is roughly constant in the case of the plate, but which increases as the square of frequency for a room. See Figure 12.17.

One may show, as in the case of the wave equation, that for a plate of stiffness parameter κ, and of surface area A, that the amount of memory required will be approximately

$$N(f_s) \propto A f_s/\kappa, \tag{12.29}$$

which, for typical plate materials and geometries ($\kappa = 0.737$, $A = 2$), and at an audio sample rate such as 48 kHz, is not particularly large (here, for a finite difference scheme, $N = 8.29 \times 10^4$). The dependence of memory requirements may be directly related to both the dispersive character

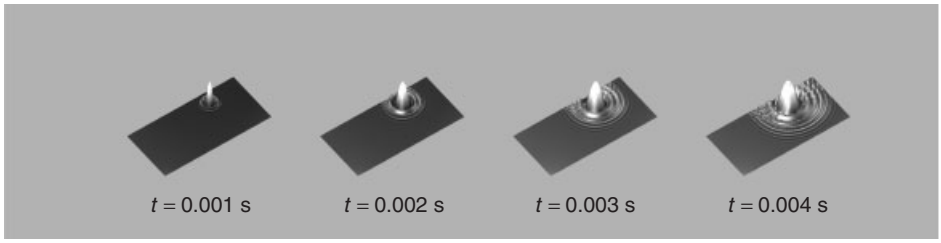

Figure 12.16 Time evolution of plate displacement, with $\kappa = 0.5$, and of aspect ratio 2, in response to a pulse, at times as indicated.

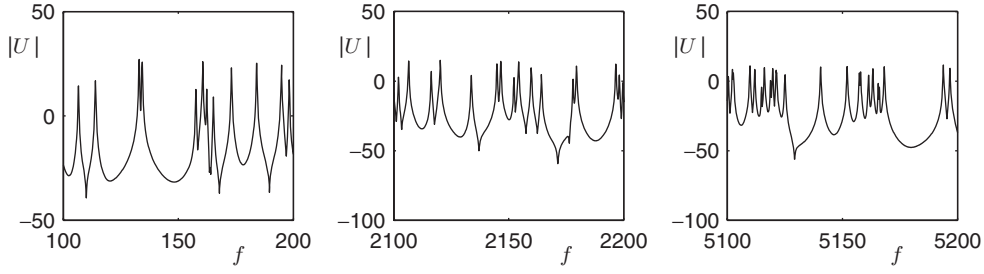

Figure 12.17 Log magnitude spectrum $|U|$ of impulse response of a plate, with $\kappa = 2$, and of aspect ratio 2, over different frequency ranges, as indicated.

of the plate, as well as the constancy of mode density [Bil09]. The operation count will again scale with $N \cdot f_s$ – which, in comparison with full 3D modeling, is in the Gflop range, which is expensive still by today's standards, but not unreasonably so. As such, it is worth looking at how one might design such a simulation through the direct time/space solution of the plate equation.

Case study: Finite difference plate reverberation

The emulation of distributed models, such as plates or springs, requires a different treatment from lumped analog components – probably the best-known distributed modeling techniques in sound synthesis are digital waveguides [Smi10], which are extremely efficient for problems in 1D, but for which such an efficiency advantage does not extend to problems in higher dimensions. Mainstream time-domain numerical simulation techniques (of which finite difference schemes [Str89] are the easiest to understand, and simplest to program) are an attractive alternative. Here, a very simple time-domain finite-difference scheme [Str89] will be presented, based on the direct solution to the equation of motion (12.25) for a thin plate. Finite difference schemes for plates are discussed by various authors (see, e.g., [Szi74]), and have been used in musical acoustics applications [LCM01], as well as in the present case of plate reverberation [Arc08, BAC06, Bil07].

Equation (12.25) describes free vibration of a lossless thin plate. A simple extension to the more realistic case of frequency-independent loss, and where the plate is driven externally (as in a reverberation unit) is

$$\frac{\partial^2 u}{\partial t^2} = -\kappa^2 \nabla^2 \nabla^2 u - 2\sigma \frac{\partial u}{\partial t} + \delta(x - x_i, y - y_i) f(t)/\rho H. \tag{12.30}$$

Here $\sigma \geq 0$ is a loss parameter – it is related to a global T_{60} reverberation time by $T_{60} = 6\ln(10)/\sigma$. More realistic (frequency-dependent) models of loss are available [CL01], but the above is sufficient

for a simple demonstration of finite difference techniques. $f(t)$ is a force signal in N, assumed known, and applied at the location $x = x_i$, $y = y_i$, where here, δ is a Dirac delta function.

The first step in the design of any audio effect is the choice of a sample rate f_s. In simulation applications, it is more natural to make use of a time step $T = 1/f_s$. Another choice which must be made is that of a spatial grid, at the points of which an approximate solution to a partial differential equation will be computed. In the case of a plate defined over a rectangular domain $x \in [0, L_x]$, and $y \in [0, L_y]$, the obvious choice is a rectangular grid, of spacing X between adjacent points. The grid function $u^n_{l,m}$ then represents an approximation to the solution $u(x, y, t)$ of (12.25), at times $t = nT$, and at locations $x = lX$, and $y = mX$, for integer n, l and m. See Figure 12.18(a). It makes sense to choose X such that the grid spacing divides the side lengths as evenly as possible, i.e., $L_x/X \approxeq N_x$, $L_y/X \approxeq N_y$ for some integers N_x and N_y. It is not possible to do this exactly for arbitrary side lengths – to remedy this, one could choose distinct grid spacings X and Y in the x and y directions, but this subtle point will not be explored further here. There are, of course, many other ways of choosing grids, which may or may not be regular – finite element methods [Coo02], for example, are usually designed to operate over unstructured grids, and are suitable for problems in irregular geometries. Consider first the first and second partial time derivatives which appear in (12.25). The simplest finite-difference approximations, δ_t and δ_{tt} may be written as

$$\delta_t u^n_{l,m} = \frac{1}{2T}\left(u^{n+1}_{l,m} - u^{n-1}_{l,m}\right) \approxeq \frac{\partial u}{\partial t}(x = lX, y = mX, t = nT) \tag{12.31}$$

$$\delta_{tt} u^n_{l,m} = \frac{1}{T^2}\left(u^{n+1}_{l,m} - 2u^n_{l,m} + u^{n-1}_{l,m}\right) \approxeq \frac{\partial^2 u}{\partial t^2}(x = lX, y = mX, t = nT). \tag{12.32}$$

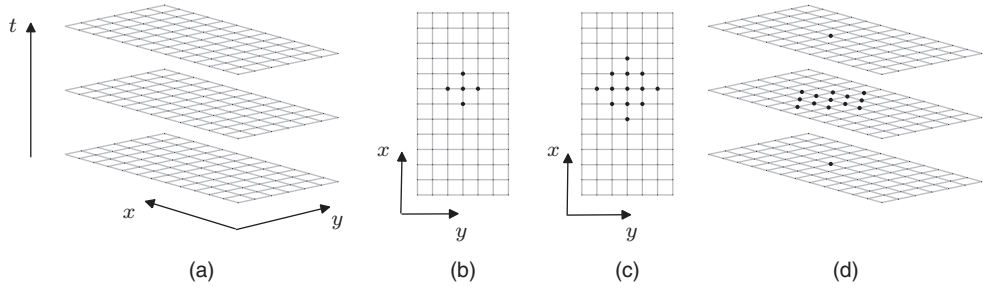

Figure 12.18 (a) Computational grid, defined over Cartesian coordinates x and y, and for time t. (b) Computational footprint of the five-point Laplacian operator, and (c) when applied twice. (d) Computational footprint for the scheme (12.34).

The other operation which must be approximated is the Laplacian ∇^2, as defined in (12.26). Here is the simplest (five-point) choice,

$$\begin{aligned}\delta_{\nabla^2} u^n_{l,m} &= \frac{1}{X^2}\left(u^n_{l+1,m} + u^n_{l-1,m} + u^n_{l,m+1} + u^n_{l,m-1} - 4u^n_{l,m}\right) \\ &\approxeq \nabla^2 u(x = lX, y = mX, t = nT),\end{aligned} \tag{12.33}$$

see Figure 12.18(b). System (12.30) requires a double application of this operator – see Figure 12.18(c). Near the edges of the domain, this operation appears to require values which lie beyond the edges of the grid – these may be set by applying the appropriate boundary conditions. For a plate reverberation unit, these will be of free type, but in the **MATLAB** example which follows, for simplicity, they have been set to be of a simply supported type (i.e., the plate is constrained at its edges, but able to pivot). A complete treatment of numerical boundary conditions is beyond the scope of the present chapter – see [Bil09] for more information.

A complete scheme for (12.25) then follows as

$$\delta_{tt} u_{l,m}^n = -\kappa^2 \delta_{\nabla^2} \delta_{\nabla^2} u_{l,m}^n - 2\sigma \delta_t u_{l,m}^n + \frac{1}{\rho H X^2} \delta_{l_i,m_i} f^n. \tag{12.34}$$

This scheme relies on a discrete delta function δ_{l_i,m_i}, picking out a grid location corresponding to $x = x_i$, and $y = y_i$ (perhaps through truncation), and a sampled input signal f^n. The scheme operates over three time levels, or steps, as shown in (d), and is also explicit – values at the unknown time step $n + 1$ may be written in terms of previously computed values (after expanding out the operator notation above) as

$$\begin{aligned}(1 + \sigma T) u_{l,m}^{n+1} = {} & 2u_{l,m}^n - (1 - \sigma T) u_{l,m}^{n-1} - \mu^2 \left(u_{l+2,m}^n + u_{l-2,m}^n + u_{l,m+2}^n + u_{l,m-2}^n \right) \\ & - 2\mu^2 \left(u_{l+1,m+1}^n + u_{l+1,m-1}^n + u_{l-1,m+1}^n + u_{l-1,m-1}^n \right) \\ & + 8\mu^2 \left(u_{l+1,m}^n + u_{l+1,m}^n + u_{l,m+1}^n + u_{l,m-1}^n \right) - 20\mu^2 u_{l,m}^n \\ & + \frac{T^2}{\rho H X^2} \delta_{l_i,m_i} f^n, \end{aligned} \tag{12.35}$$

where $\mu = \kappa T / X^2$ is a numerical parameter for the scheme. It may be shown that for stability, one must choose

$$\mu \leq 1/4 \tag{12.36}$$

and in fact, in order to reduce artifacts due to numerical dispersion (i.e., additional unwanted dispersive behavior induced by the scheme itself, on top of that inherent to the plate itself), this bound should be satisfied as near to equality as possible. In audio applications, this allows the grid spacing X to be chosen in terms of the sample rate. See [Bil09] for much more on the properties of this scheme.

In implementation, it can be useful to rewrite this scheme in a vector-matrix form. A grid function $u_{l,m}^n$, representing the values of a grid function over a rectangular region, may be rewritten as a column vector $\mathbf{u}^n$, consisting of consecutive "stacked" columns of values over the grid, at time step n. Because the scheme as a whole is linear, and explicit, it must then be possible to rewrite it as the two-step recursion,

$$\mathbf{u}^{n+1} = \mathbf{B}\mathbf{u}^n - \mathbf{C}\mathbf{u}^{n-1} + \mathbf{r} f^n. \tag{12.37}$$

Here, the effects of loss and the spatial difference operators have been consolidated in the (extremely sparse) matrices $\mathbf{B}$ and $\mathbf{C}$, and $\mathbf{r}$ is a column vector which chooses an input location. Output may be drawn from the scheme at a location $x = x_o$, $y = y_o$ by simply reading u_{l_o,m_o}^n at each time step. In vector form, the output y^n may be written as

$$y^n = \mathbf{q}^T \mathbf{u}^n \tag{12.38}$$

for some vector $\mathbf{q}$, which, in the simplest case, will consist of a single value at the readout location.

Below is a very simple **MATLAB** implementation of the plate reverberation algorithm mentioned above – it generates a reverberant output, for a given plate, sample rate, and excitation and readout points. It's not particularly fast (at least in **MATLAB**) – but not extremely slow either!

M-file 12.8 (`platerevsimp.m`)

```
% platerevsimp.m
% Authors: Välimäki, Bilbao, Smith, Abel, Pakarinen, Berners
% global parameters
```

```
[f, SR] = wavread('bassoon.wav');   % read input soundfile and sample rate
rho = 7850;                         % mass density (kg/m^3)
E = 2e11;                           % Young's modulus (Pa)
nu = 0.3;                           % Poisson's ratio (nondimensional)
H = 0.0005;                         % thickness (m)
Lx = 1;                             % plate side length---x (m)
Ly = 2;                             % plate side length---y (m)
T60 = 8;                            % 60 dB decay time (s)
TF = 1;                             % extra duration of simulation (s)
ep = [0.5 0.4];                     % center location ([0-1,0-1]) nondim.
rp = [0.3 0.7];                     % position of readout([0-1,0-1]) nondim.
% derived parameters
K_over_A = sqrt(E*H^2/(12*rho*(1-nu^2)))/(Lx*Ly);  % stiffness parameter/area
epsilon = Lx/Ly;                     % domain aspect ratio
T = 1/SR;                            % time step
NF = floor(SR*TF)+size(f,1);         % total duration of simulation (samples)
sigma = 6*log(10)/T60;               % loss parameter
% stability condition/scheme parameters
X = 2*sqrt(K_over_A*T);             % find grid spacing, from stability cond.
Nx = floor(sqrt(epsilon)/X);        % number of x-subdivisions/spatial domain
Ny = floor(1/(sqrt(epsilon)*X));    % number of y-subdivisions/spatial domain
X = sqrt(epsilon)/Nx;               % reset grid spacing
ss = (Nx-1)*(Ny-1);                 % total grid size
% generate scheme matrices
Dxx = sparse(toeplitz([-2;1;zeros(Nx-3,1)]));
Dyy = sparse(toeplitz([-2;1;zeros(Ny-3,1)]));
D = kron(speye(Nx-1), Dyy)+kron(Dxx, speye(Ny-1)); DD = D*D/X^4;
B = sparse((2*speye(ss)-K_over_A^2*T^2*DD)/(1+sigma*T));
C = ((1-sigma*T)/(1+sigma*T))*speye(ss);
% generate I/O vectors
rp_index = (Ny-1)*floor(rp(1)*Nx)+floor(rp(2)*Ny);
ep_index = (Ny-1)*floor(ep(1)*Nx)+floor(ep(2)*Ny);
r = sparse(ss,1); r(ep_index) = T^2/(X^2*rho*H);
q = sparse(ss,1); q(rp_index) = 1;
% initialize state variables and input/output
u = zeros(ss,1); u2 = u; u1 = u;
f = [f;zeros(NF-size(f,1),1)];
out = zeros(NF,1);
% main loop
for n=1:NF
    u = B*u1-C*u2+r*f(n);   % difference scheme calculation
    out(n) = q'*u;          % output
    u2 = u1; u1 = u;        % shift data
end
% play input and output
soundsc(f,SR); pause(2); soundsc(out,SR);
```

This example is written to be compact and readable – and many more realistic features which have been omitted could be added in. Among these are:

- Frequency-dependent damping – at the moment the T_{60} is set to be uniform at all frequencies. This could be done directly in the PDE model above, or possibly through introducing a model of a damping plate.
- Boundary conditions are set to be simply supported – allowing the easy generation of difference matrices. These should ideally be set to be free, or going further, to incorporate the effect of point supports.

- Only a single output is generated – multiple outputs (such as stereo, but perhaps more in a virtual environment) should be allowed. Notice that multiple outputs may be generated essentially "for free," as the entire state (**u**) is available.
- Output displacement is taken; in real plate reverberation units, it is rather an acceleration – acceleration can be derived from displacement easily through a double time difference operation.
- Post-processing (analog electronic) and the behavior of the pickup are not modelled.

Information on all these extensions appears in various publications, including those mentioned at the beginning of this section.

12.4.3 Springs and spring reverberation

A helical spring under low tension has long been used as an artificial reverberation device [Ham41, YI63, Sta48]. In most units, the spring is driven through an electromagnetic coupling at one end, and readout is taken via a pickup at the opposite end. Spring reverbs come in a variety of sizes and configurations, sometimes involving multiple springs. A typical set-up is shown in Figure 12.19(a).

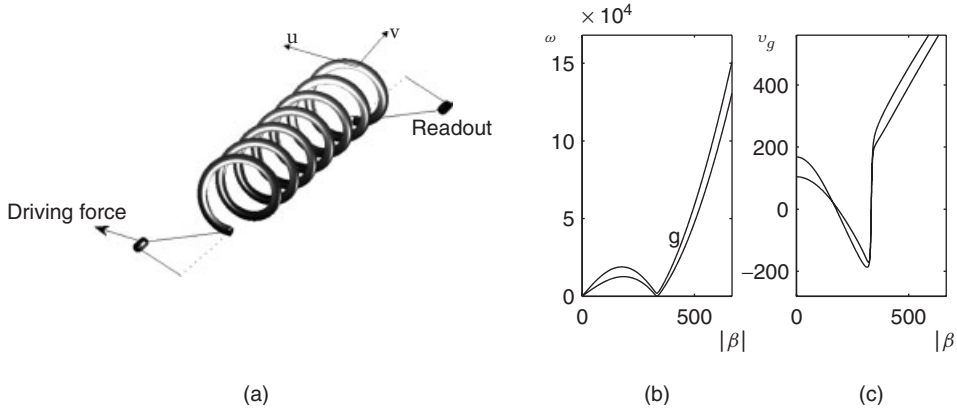

Figure 12.19 (a) A typical spring reverberation configuration, for a spring driven at one end, and with readout taken from the other; longitudinal and transverse motion of the spring are indicated by u and v. (b) Dispersion relations $\omega(\beta)$, and (c) group velocities $v_g(\beta)$.

The physics of helical structures is far more involved than that of the flat plate – see [Wit66, DdV82] for typical models. The important point is that the various types of motion (transverse to the spring, in both polarizations, and longitudinal) are strongly coupled in a way that they are not for a straight wire. A complex combination of relatively coherent wave propagation (leading to echoes) and highly dispersive propagation is present, leading to a characteristic sound which resembles both real room responses, and plate reverbs.

One can garner much useful information from the dispersion relations for a spring, as shown in Figure 12.19(b). For springs of dimensions and composition as found in reverberation devices, there are two relations which lie within the audio range – both correspond to mixtures (wavenumber-dependent!) of longitudinal and transverse motion. Both curves exhibit a "hump" in the low wavenumber range corresponding to a cutoff – for most spring reverb units, this occurs in the range between 2 kHz and 5 kHz. An examination of the group velocities, as shown in Figure 12.19(c) allows for some insight with regard to the behavior of spring responses, an example of which is shown at top in Figure 12.20. In the range of low wavenumbers, the group velocities approach a

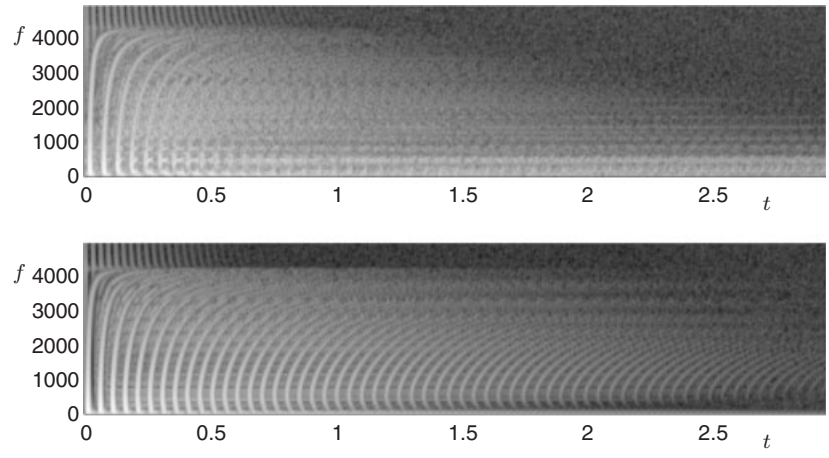

Figure 12.20 Top, spectrogram of a measured spring reverberation response, and bottom, that of a synthetic response produced using a finite difference model.

constant value, and thus dispersion is low at low frequencies, and strong echoes are present. But as the wavenumber increases towards the cutoff, the group velocity decreases, and thus the frequency components of the echoes are distorted increasingly (slowed) near the cutoff. Above the cutoff, the group velocity is much faster, and leads to distorted echoes which recur at a higher rate. Above the cutoff, the response is very similar to transverse motion for a straight bar.

Emulation techniques

As in the case of plate reverberation, there is a variety of techniques available – and as of 2010, physical modeling of springs is still an open research problem. The most rigorous approach, given a partial differential equation model, is again a time-stepping method such as finite differences, which has been used by this author [Bil09, BP10], as well as Parker [Par08] in order to generate synthetic responses. See Figure 12.20(b) for a comparison between a measured response and synthetic output. This approach turns out to be rather expensive, and given that the system is essentially 1D (i.e., for a thin spring, the equations of motion may be written with respect to a single coordinate running along the spring helix), structures based on delay lines and allpass networks are a possibility, and an excellent compromise between physical and perceptual modeling. See, e.g., [ABCS] and [VPA10].

The difficulties here lie in associating such a model with an underlying model in a definitive way. Another approach, as yet unexplored, could be based on the use of a modal decomposition. In other words, given appropriate boundary conditions, one may solve for the modes of vibration of the helix. This is the approach taken in the analysis of helical structures in mainstream applications – see, e.g., [LT01].

12.5 Tape-based echo simulation

12.5.1 Introduction

In this section, we turn our attention to simulating tape echo, focusing on two popular units, the Roland Space Echo, used widely in dub and reggae music, and the Maestro Echoplex, popular among guitarists.

The Echoplex was first released in 1959. Four versions were made, the EP-1 and EP-2 employing tube circuitry, and the EP-3 and EP-4 using solid-state electronics. An EP-4 is shown in

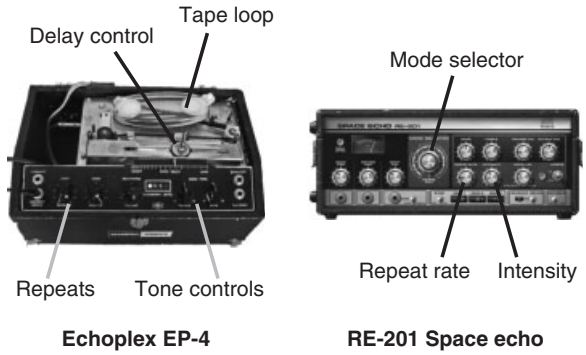

Figure 12.21 Maestro Echoplex EP-4 (left) and Roland RE-201 Space Echo (right).

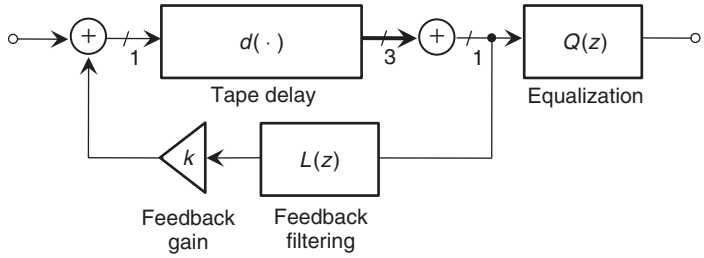

Figure 12.22 Tape Delay Signal Flow Architecture. The Space Echo provides up to three separate delays which are summed (as shown); the Echoplex features a single delay.

Figure 12.21, and its signal flow architecture in Figure 12.22. Playback and erase heads are fixed on either side of a slot along which the record head is free to move. The record head writes to the moving tape, and the recorded signal is later read as the tape passes by the playback head. Moving the delay control changes the distance between the record and playback heads, and, therefore, the time delay between input and output. With a nominal tape speed of 8 ips, delays roughly in the range of 60 ms to 600 ms are available.

As seen in Figure 12.22, the input signal is delayed and fed back, with the delayed output equalized according to the tone controls. The amount of feedback is controlled by the "Repeats" knob, producing a loop gain k roughly in the range $k \in [0, 2]$. For loop gains less than one, the Echoplex produces a decaying series of echoes. With a loop gain greater than one, sound present in the system will increase in amplitude until saturating, and take on the bandpass spectral quality of the loop filtering, with a rhythmic temporal character determined by the delay setting. When $k > 1$, no input signal is needed to produce an output, as tape hiss and 60 Hz harmonics present in the system will cause the unit to self-oscillate.

The movable record head is capable of producing extreme Doppler shifts [AAS08]. If the delay control is quickly moved away from the playback head, the input may be Doppler shifted down sufficiently that the bias signal appears in the audio band. If the delay control is moved toward the playback head at a rate faster than the tape speed, a high-bandwidth transient will be recorded as the record head slows and is overtaken by the tape.

The Space Echo, released in 1973, is shown in Figure 12.21. It has similar controls to the Echoplex, but uses a variable tape speed controlled by its "Repeat Rate" knob to fix the time delay between the record and playback heads. As shown in Figure 12.23, the Space Echo has three playback heads whose outputs are summed according to the "Mode Selector" to form the delayed output.

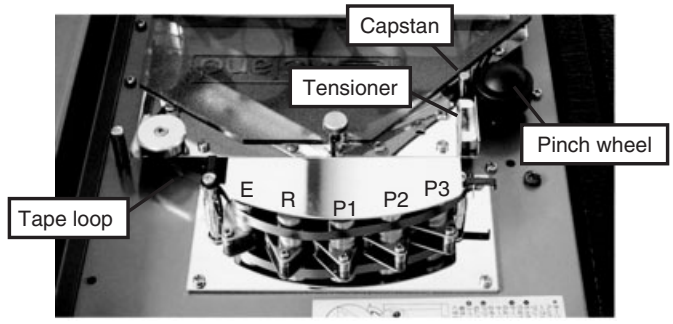

Figure 12.23 Space Echo Tape Transport. The erase (E), record (R), and three playback heads (P1, P2, P3) are shown.

Like the Echoplex, the Space Echo also has a saturating signal path, and a feedback loop gain k ranging from $k \in [0, 2]$ according to "Intensity" control. Noise in the electronics will produce a self-oscillation similar to that of the Echoplex.

The Space Echo and Echoplex may be modeled according to their signal flow architecture, as shown in Figure 12.22. Unlike digital or electronic delays, the mechanics of the tape delay transport produce subtle fluctuations in the time delay which add to the character of the effect. In Section 12.5.2 below, the mechanics of the tape transport are described, and methods for simulating the time-delay trajectory are presented. In Section 12.5.3, details of the signal path are noted, including tape and electronic saturation, and an analysis of the tone control circuit.

12.5.2 Tape transport

Delay trajectories

The tape transport mechanism employed by the Space Echo and Echoplex uses a metal capstan engaged with a rubber pinch wheel to pull the tape past the record and playback heads. As seen in Figure 12.23, the tape is spliced into a loop, and passes through a tensioner before arriving at the pinch wheel and re-entering its cartridge. The various forces acting on the tape produce a tape speed which varies somewhat over time. This varying tape speed results in a fluctuating time delay, which adds to the character of the delayed signal and is important to the time evolution of the sound produced when the feedback gain is greater than 1.0. It should be pointed out that as the tape splice passes by the playback head, a modest dropout is produced.

Figure 12.24 shows the time evolution of the time delays provided by the Space Echo playback head P1 measured at a variety of "Repeat Rate" (i.e., tape speed) settings. The resulting average time delays range from a little over 60 ms to a little over 160 ms. Figure 12.24 also shows the same delay trajectories with their means subtracted and offset for display. Both quasi-periodic and noise-like components are evident. Figure 12.25 shows spectra of the time-delay trajectories of Figure 12.24, plotted on frequency axes normalized according to their respective time delays. The spectra peak at the capstan rotation rate (about 22 Hz for the shortest time delay) and at the pinch-wheel rotation rate (near 3.5 Hz for the shortest time delay).

There are a series of nulls appearing near integer multiples of 15 Hz, and an overall low pass quality to the spectra. This pattern is reminiscent of a sinc function corresponding to a time-domain integration over a 60 ms window.

The 60 ms window length is precisely the (normalized) time delay, and represents the integration of velocity fluctuations as the tape travels between the record and playback heads. Tape speed disturbances completing an integer number of periods (slowing down and speeding up the tape)

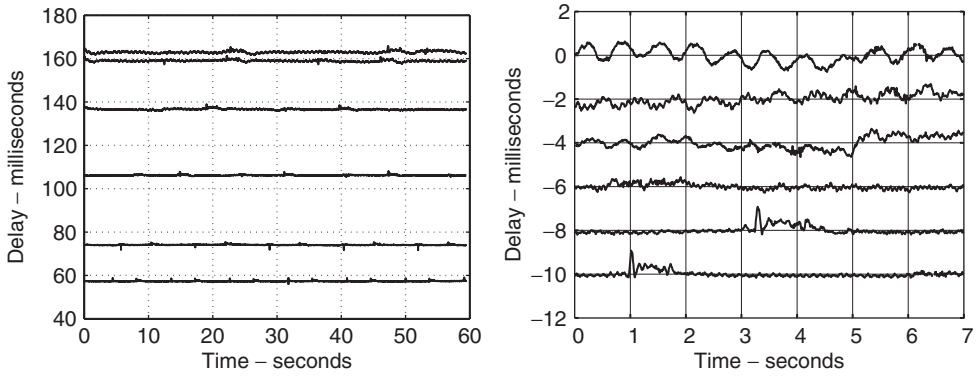

Figure 12.24 Space Echo delay trajectories (left), mean-subtracted trajectories, (offset, right).

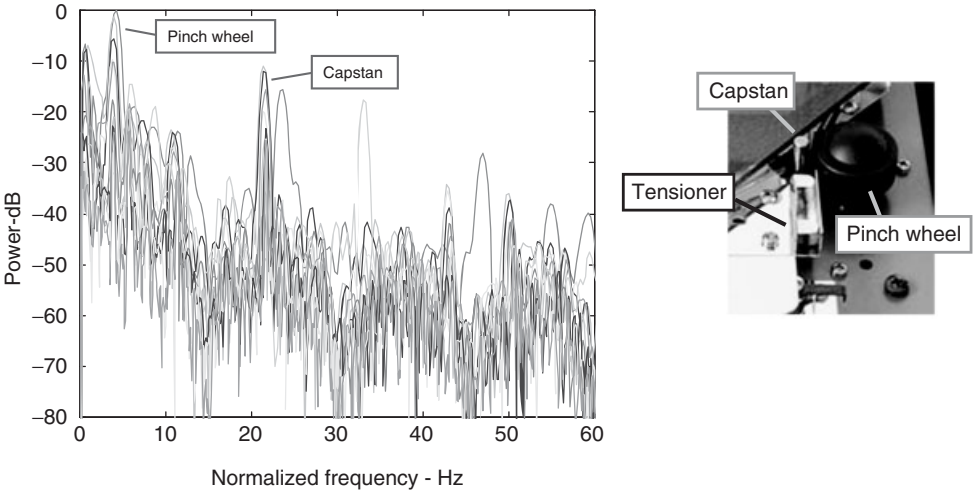

Figure 12.25 Delay trajectory spectra. Spectra of the delay trajectories of Figure 12.24 are plotted on frequency axes scaled to produce a 60 ms nominal delay.

while traveling between the heads will have no effect on the resulting time delay, and therefore, do not appear in the time delay spectra.[5]

Another effect of the integration can be seen in Figure 12.24. Note that the longer the time delay, the longer the integration, and the greater the time-delay fluctuations. In fact, the amplitude of time-delay fluctuations is roughly proportional to the average delay.

In Figure 12.24, transients appear in the two time-delay trajectories offset at −10 ms and −8 ms. These temporary increases in time delay are the result of the tape splice squeezing through the tensioner and then the pinch wheel. The tape is temporarily slowed, and the time delay later temporarily increased. The passage of the splice – about once every nine seconds for the fast tape speed setting, and about every 27 seconds for the slow tape speed setting – is important for creating an evolving sound in the presence of heavy feedback.

[5] As an aside, note that if the distance between the record head and the playback head were equal to integer multiples of the capstan and pinch-wheel circumferences, the quasi-periodic time delay trajectory components would be eliminated.

Other factors influence the time-delay trajectory. Compared to the trajectories seen in Figure 12.24, measured using a new tape, the fluctuations when using old tape have a much larger stochastic component.

Delay trajectory physical model

The delay trajectory may be generated using a model of the tape transport mechanics. The angular rotation rate of the capstan ω_c is changed according to its moment of inertia I and applied torques,

$$I\frac{\mathrm{d}\omega_c}{\mathrm{d}t} = \tau(r) - \mu T \rho_c - \gamma(\theta_p). \tag{12.39}$$

In the above expression, $\tau(r)$ represents the motor torque applied as a function of the Space Echo "Repeat Rate" setting r ($\tau(r)$ is presumed constant for the Echoplex). The term $\mu T \rho_c$ is the torque resulting from the tensioner friction – it is the tape tension T, scaled by the coefficient of friction μ, acting at the capstan radius ρ_c. Rather than applying a constant torque according to $\mu T \rho_c$, a unit-energy noise process is suggested, as this corresponds to the stochastic nature of frictional forces [Ser04].

The torque $\gamma(\theta_p)$ accounts for angular velocity changes due to the periodic deformation of the pinch wheel as it rotates about its axis. The distance between the capstan center and pinch-wheel axle is fixed, and any irregularities in pinch-wheel radius will result in corresponding deformations in the pinch-wheel rubber. Assuming a spring constant k, the energy stored in the pinch-wheel deformation at pinch-wheel angle θ_p is

$$E_s = \frac{1}{2}k\rho_p^2(\theta_p), \tag{12.40}$$

where $\rho_p(\theta_p)$ is the pinch-wheel radius at pinch-wheel rotation angle θ_p. In a time period Δt, the energy stored is

$$\Delta E_s = k\rho_p(\theta_p)\frac{\mathrm{d}\rho_p}{\mathrm{d}\theta_p}\frac{\mathrm{d}\theta_p}{\mathrm{d}t}\cdot\Delta t. \tag{12.41}$$

The kinetic energy change ΔE_ω over the same time period is

$$\Delta E_\omega = I\omega_c\frac{\mathrm{d}\omega_c}{\mathrm{d}t}\cdot\Delta t. \tag{12.42}$$

The pinch-wheel deformation results in a torque that balances the energy changes, $\Delta E_s + \Delta E_\omega = 0$,

$$\gamma(\theta_p) = I\frac{\mathrm{d}\omega_c}{\mathrm{d}t} = -k\rho_c\frac{\mathrm{d}\rho_p}{\mathrm{d}\theta_p}, \tag{12.43}$$

where the ratio of the capstan and pinch-wheel radii was substituted for the inverse ratio of their angular velocities. A source of pinch-wheel deformation is an off-center axle, generating a pinch-wheel radius

$$\rho_p(\theta_p) = \rho_0\left[(\cos\theta_p - \beta)^2 + \sin^2\theta\right]^{\frac{1}{2}}, \tag{12.44}$$

where ρ_0 is the pinch-wheel radius, and β is the distance of the pinch-wheel axle from the pinch-wheel center. This pinch-wheel radius generates the sensitivity

$$\frac{\mathrm{d}\rho_p}{\mathrm{d}\theta_p} = \frac{\rho_0^2\beta\sin\theta_p}{\rho_p(\theta_p)}. \tag{12.45}$$

The time delay $\delta(t)$ between the record and playback heads may be found by integrating the tape speed $v(t)$ until the record head-playback head distance D is achieved,

$$D = \int_{t}^{t+\delta(t)} v(\tau)\mathrm{d}\tau, \qquad v(t) = \rho_c \frac{\mathrm{d}\theta_c}{\mathrm{d}t}, \tag{12.46}$$

where $v(t)$ is the tape speed produced by the spinning capstan. Compared to the global average tape speed v_0, the tape speed fluctuations $v(t) - v_0$ are small, and the varying time delay may be approximated by

$$\delta(t) \approx \delta_0 - \int_{t}^{t+\delta_0} \frac{v(\tau) - v_0}{v_0}\mathrm{d}\tau, \tag{12.47}$$

where δ_0 is the nominal delay, $\delta_0 = v_0/D$. The time-delay trajectory is therefore the difference between the nominal delay and a running mean of the tape speed fluctuations.

Delay trajectories synthesized using (12.39) and (12.47) are shown in Figure 12.26, along with the corresponding delay fluctuations. A simple tape splice model was included.

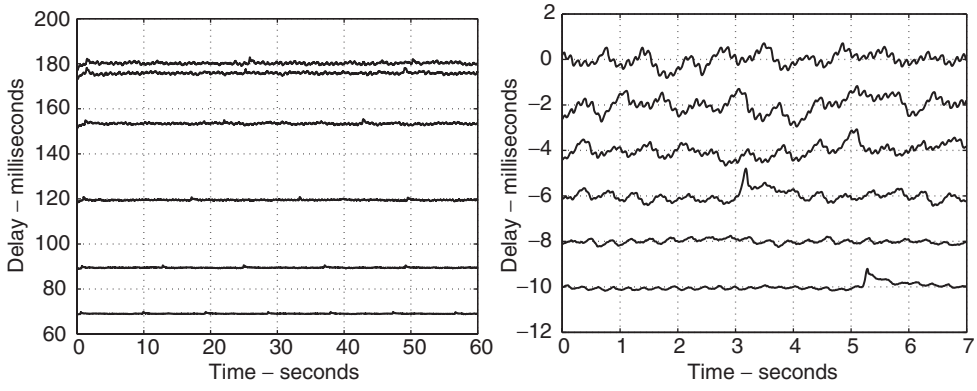

Figure 12.26 Space Echo synthesized delay trajectories (left), mean-subtracted (offset, right).

Delay-trajectory signal model

The fluctuating time delay driving the tape-delay model may be formed without reference to the mechanics of the tape transport. Instead, the observed time-delay trajectory characteristics may be reproduced via a "signal model."

In this approach, the tape delay is formed, as shown in Figure 12.27, by adding the three time-delay trajectory components: a mean delay, as determined by the delay control position for the Echoplex or the "Repeat Rate" for the Space Echo; quasi-periodic processes representing the observed capstan and pinch-wheel harmonics; and a stochastic low-frequency drift. This sum is filtered via a running mean having length equal to the time delay. The pinch-wheel and capstan components are generated by summing sinusoids at integer multiples of the observed capstan and pinch-wheel rotation rates, with amplitudes equal to those seen in the measured time-delay fluctuation spectra. The sinusoid phases are incremented according to their respective measured frequencies, with small low pass, zero-mean noise processes added to the sinusoid phases to introduce observed frequency fluctuations. The low-frequency drift component is formed by filtering white Gaussian noise.

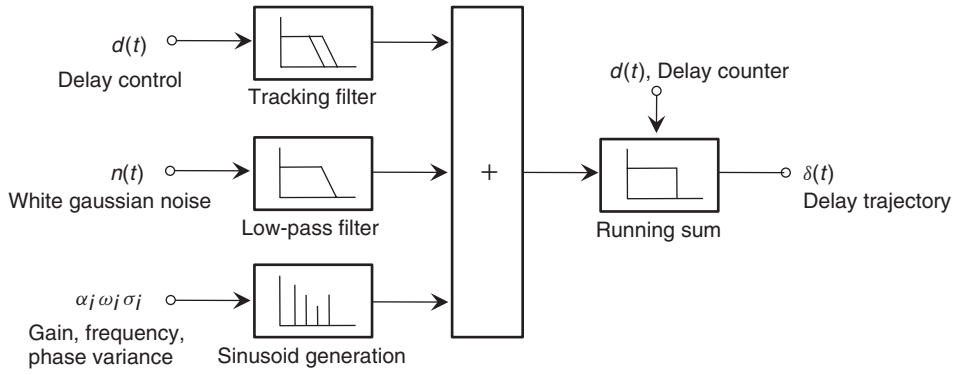

Figure 12.27 Delay trajectory signal model.

Delay control ballistics

As described above, the Space Echo time delay is controlled by adjusting the speed of the motor driving the capstan-pinch wheel assembly; the Echoplex time delay is controlled directly by positioning the record head. In both of these controls, there is a time lag between when the control is moved and when the desired delay is reached. This is seen in Figure 12.28 for the case of the Space Echo having its "Repeat Rate" control quickly adjusted between its slowest and fastest settings.

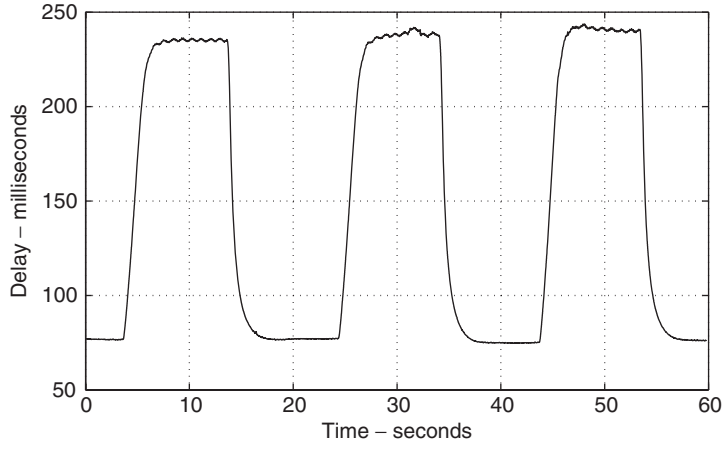

Figure 12.28 Space Echo delay trajectory, stepped "Repeats Rate" control.

The behavior seen in Figure 12.28 is well approximated by a so-called leaky integrator: a target signal v_T is tracked to produce $v(n)$ using a first-order update,

$$v(n) = (1 - \lambda)v_T + \lambda v(n - 1), \tag{12.48}$$

where λ, the "forgetting factor," governs the rate at which the target signal v_T replaces the current estimate $v(n)$. With λ near 0, the target v_T quickly replaces the integrator output $v(n)$, whereas with λ near 1, the integrator output slowly relaxes to the target. Given a time constant τ and sampling rate f_s, the forgetting factor

$$\lambda = \mathrm{e}^{-1/\tau f_s} \tag{12.49}$$

will produce a leaky integrator, whose output will be a factor $1 - 1/e$ closer to a constant target signal after a period of time τ.

Referring again to Figure 12.28, the tape speed is seen to relax to its target value with a time constant on the order of 1 second when the target tape speed is greater than the current tape speed. However, when the target tape speed is less than the current tape speed, the time constant is slower at about 2 seconds. This behavior may be captured using different forgetting factors, depending on whether the target tape speed is greater than or less than the current tape speed.

Time-varying delay implementation

For arbitrary interpolated delay-line access, it is convenient to use Lagrange interpolation, since the needed filter coefficients can be calculated inexpensively for any fractional delay [Laa96]. The Lagrange interpolation filter is defined as the FIR filter which models the ideal fractional delay $e^{-j\delta\omega}$ with a maximally flat approximation at DC. In other words, for a fractional delay of δ, the Nth-order Lagrange filter will have a frequency response $H(e^{j\omega})$ which matches $e^{-j\delta\omega}$ and its first N derivatives at $\omega = 0$. Usually, the M-tap Lagrange filter will be used to produce a delay δ such that $M/2 < \delta < M/2 + 1$ for even M.

The ideal fractional delay $e^{-j\delta\omega}$ has nth derivative $(-j)^n\delta^n$ at $\omega = 0$. The Nth-order FIR filter has a frequency response

$$H(e^{j\omega}) = \sum_{n=0}^{N} b_n e^{-j\omega n} \tag{12.50}$$

which has nth derivative

$$(-j)^n b_1 + 2^n(-j)^n b_2 + \cdots + N^n(-j)^n b_n \tag{12.51}$$

at $\omega = 0$. Matching the response and first N derivatives of $H(e^{j\omega})$ at DC to those of the ideal fractional delay defines the filter coefficients b_n for the desired delay δ,

$$\mathbf{V}\vec{b} = \vec{\delta}, \tag{12.52}$$

where $\mathbf{V}$ is the Vandermonde matrix defined by the row $[0 \quad 1 \quad 2 \quad \cdots \quad N]$, $\vec{b}$ is the column of filter coefficients $[b_0 \quad b_1 \quad b_2 \quad \cdots \quad b_N]^T$, and $\vec{\delta} = [1 \quad \delta \quad \delta^2 \quad \cdots \quad \delta^N]^T$. For $N = 3$, we have

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 4 & 9 \\ 0 & 1 & 8 & 27 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 \\ \delta \\ \delta^2 \\ \delta^3 \end{bmatrix}. \tag{12.53}$$

The filter coefficients $\vec{b}$ can be found by inverting $\mathbf{V}$,

$$\vec{b} = \mathbf{V}^{-1}\vec{\delta}. \tag{12.54}$$

For real-time operation, the matrix inversion can be carried out in advance. The product $\mathbf{V}^{-1}\vec{\delta}$ allows efficient calculation of b_n for a given fractional delay δ. As the filter order N increases, the Lagrange interpolator coefficients converge towards a sampled sinc function, which would give ideal bandlimited interpolation.

M-file 12.9 (`lagr2.m`)

```
% lagr2.m
% Authors: Välimäki, Bilbao, Smith, Abel, Pakarinen, Berners
```

```
% lagr.m -- Lagrange filter design script
fs=44100;
numtaps=8;   %% Make this even, please.
ordvec=[0:numtaps-1]';
VA=flipud(vander(ordvec)');
iVA=inv(VA); %matrix to compute coefs
numplots=16;
for ind=1:numplots
    eta=(ind-1)/(numplots-1);  %% fractional delay
    delay=eta+numtaps/2-1;  %% keep fract. delay between two center taps
    deltavec=(delay*ones(numtaps,1)).^ordvec;
    b=iVA*deltavec;
    [H,w]=freqz(b,1,2000);
    figure(1)
    subplot(2,1,1)
    plot(w*fs/(2*pi),(abs(H)))
    grid on
    hold on
    xlabel('Freq, Hz')
    ylabel('magnitude')
    axis([0 fs/2 0 1.1])
    subplot(2,1,2)
    plot(w*fs/(2*pi),180/pi*unwrap(angle(H)))
    grid on
    hold on
    xlabel('Freq, Hz')
    ylabel('phase')
    xlim([0 fs/2])
end
figure(1)
subplot(2,1,1);hold off;
title('Lagrange Interpolator')
subplot(2,1,2);hold off;
```

12.5.3 Signal path

Recording with AC bias

Magnetic data storage is achieved by exposing magnetic material to an external magnetic field. If the applied field is of sufficient magnitude, residual magnetization of the material will persist after the external field is removed. For a time-varying field applied along a single axis to a magnetic material, the relationship between the applied field and the magnetization of the material will take the hysteretic form shown in Figure 12.29. A demagnetized sample will follow trajectory '1' in the figure if an increasing magnetic field is applied. The magnetization will grow to a maximum M_s for large fields, where M_s is the saturation magnetization for the material. As the applied field is reduced, the magnetization will decrease along trajectory '2' until, at an applied field of zero, the magnetization will be M_r, the remanent magnetization for the material. When the applied field reaches the coercive force $-H_c$, the total magnetization of the sample will once again be zero. If the applied field becomes more negative, the sample will continue to follow trajectory '2'. If, at a later time, the applied field is once again increased, the sample will follow trajectory '3'.

The hysteretic process of Figure 12.29 leads to a highly nonlinear relationship between the applied field and the resultant magnetization. In order to linearize the recording process, an AC bias signal consisting of a high-frequency sinusoid is typically added to the audio signal. The sum of the two signals is passed to a record head which creates a localized magnetic field that is proportional to the current passing through it.

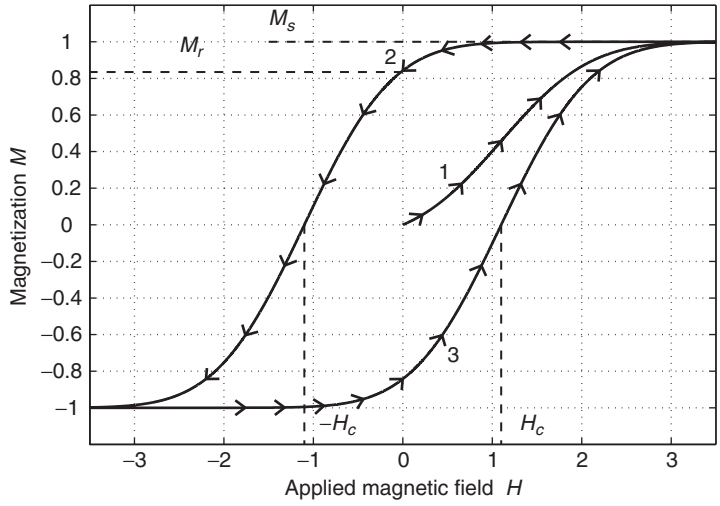

Figure 12.29 Hysteretic magnetization curve, showing saturation magnetization M_s, remanent magnetization M_r, and coercive force H_c.

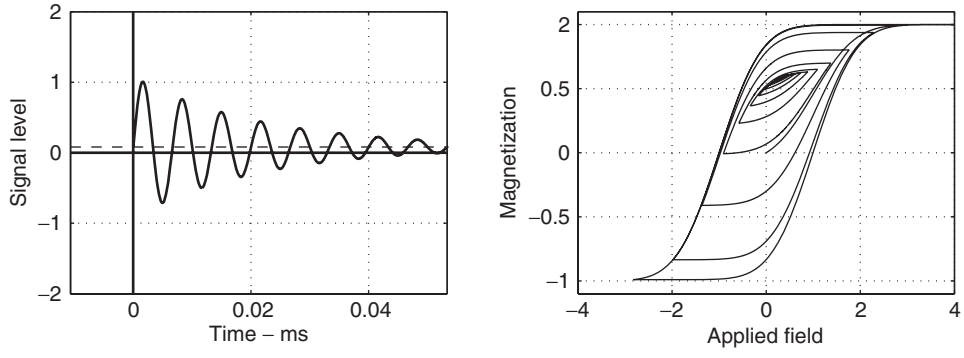

Figure 12.30 Anhysteretic recording. Decaying AC bias signal plus DC offset (left). Magnetization trajectory for decaying signal (right).

When an element of tape moves past the record head, it is exposed to the magnetic field created by the head. As the tape recedes from the head gap, the field applied to the tape becomes progressively weaker through distance, and the tape element experiences a decaying, oscillating magnetic field due to the AC bias. If the audio signal is bandlimited to frequencies much lower than the AC bias frequency, it can be considered quasistatic during the decay of the bias field. For a decaying AC bias signal added to a quasi-DC signal, the field applied to the tape would be as shown on the left of Figure 12.30. This signal produces the magnetization trajectory shown on the right-hand side of the figure. The trajectory spirals inward to a final magnetization which is largely insensitive to the details of the bias signal, and depends only upon the value of the quasi-DC component. In this way, the hysteretic behavior is eliminated, resulting in what is known as the anhysteretic recording process.

In reality, the audio component of the record signal will decay along with the bias signal as the tape element moves out of the recording head gap, producing the signal and magnetization trajectory of Figure 12.31. This is known as the modified anhysteretic recording process [Mal87].

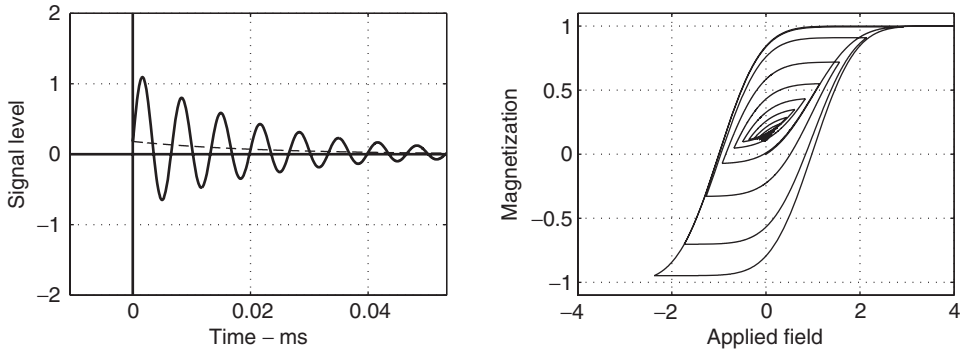

Figure 12.31 Modified anhysteretic recording. Decaying AC bias signal and signal (left). Magnetization trajectory for decaying signal (right).

With the hysteresis eliminated, the record process can be modeled as a memoryless nonlinearity. With proper biasing levels, recordings can be made with very little distortion, as long as levels are kept below M_s. The anhysteretic recording nonlinearity can be approximated by the error function $M = \text{erf}(H)$, shown in Figure 12.32.

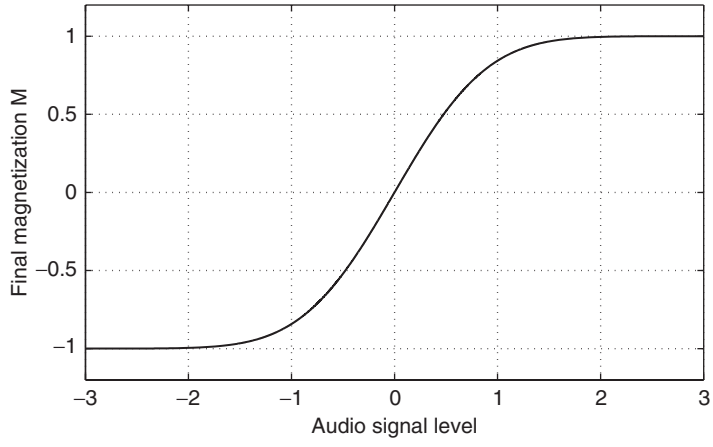

Figure 12.32 Memoryless nonlinearity for anhysteretic recording.

Tape-speed-dependent equalization

For signal reproduction, the tape is dragged across a gapped play-head which detects changes in magnetization as the tape goes by. The head is most sensitive to the space immediately surrounding the gap, but also responds to more distant elements of the tape. The recovered signal can be calculated as the derivative in time of a weighted volume integral over the space surrounding the head. The sensitivity of the head depends upon field direction as well as spatial position. This is reflected by taking a dot product between the tape magnetization and a vector representing the head's (directional) sensitivity as a function of space. The recovered signal s can be expressed as

$$s(t) = \frac{\mathrm{d}}{\mathrm{d}t} \int \vec{M} \cdot \vec{w} \; \mathrm{d}x \, \mathrm{d}y \, \mathrm{d}z \tag{12.55}$$

where $\vec{w}(x, y, z)$ represents the head's sensitivity.

The spatial integration performed by the playback head leads to wavelength-dependent equalization. At high frequencies, the wavelength recorded on tape becomes comparable to the gap width in the head, leading to a sinc-like frequency response; when an integral number of wavelengths fits in the gap, the response is diminished. This effect is called gap loss and takes the form

$$\gamma(k) = \frac{\sin(kg)}{kg} \tag{12.56}$$

where $\gamma(k)$ is the wavelength-dependent gain, k is the wavenumber, and g is the width of the head gap. A related effect happens at low frequencies when the wavelength becomes comparable to the dimension of the head core. This is called the contour effect, and produces what is known as head bump. Both gap loss and the contour effect are influenced by tape speed v, since $k \propto 1/v$. For the Roland Space Echo, since delay times are adjusted by changing the tape speed, the equalization will depend on delay. For the Echoplex, the tape speed is relatively constant, and the equalization will not be affected by delay setting.

Electronics

Equalization is applied electronically before writing to tape, and again after reading back from tape. This is done to maximize the tape's dynamic range, and to minimize the spectral coloration associated with the record/playback process. The equalization at the record head can be measured directly if the AC bias source is temporarily disconnected. The record head itself presents an inductive load over most of the audio band. At low frequencies, the series resistance of the head windings can become dominant. The field applied to tape is proportional to the current through the head, which can be measured directly or calculated from the head voltage if the head impedance is known. In many cases the high-frequency content of the signal is emphasized at the record head. It should be noted that at some frequencies the amplifier circuit will clip before reaching the level required to saturate the tape. This can be observed with both the Space Echo and the Echoplex.

After playback, the signal is equalized to give the desired transfer function, as measured from input to output. For the Space Echo and Echoplex, the overall equalization is relatively flat, and bandlimited. Plots of the equalization as a function of tape speed for the Space Echo are shown in Figure 12.33.

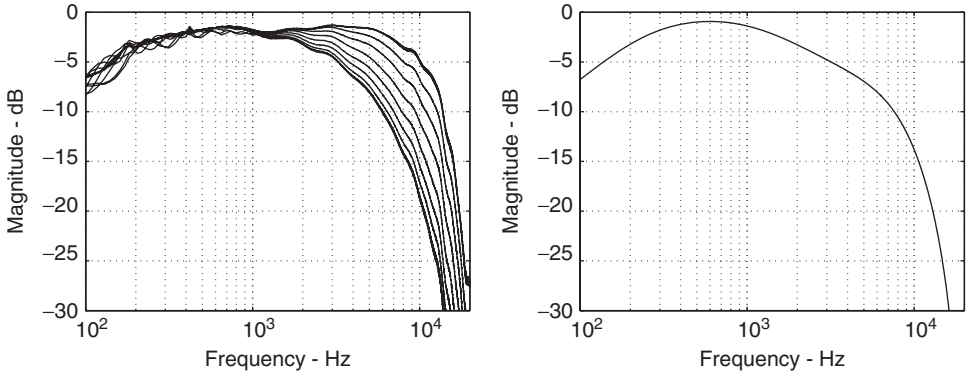

Figure 12.33 Space Echo equalization, various tape speeds (left), Echoplex loop equalization (right).

When in feedback mode, the Space Echo and Echoplex have an overall equalization associated with one trip around the feedback loop. This equalization can be measured by taking spectral ratios between successive echoes of a broadband signal. The loop equalization is the concatenation of

parts of the pre-record and post-playback equalization, along with whatever equalization is present in the feedback path. The loop equalization for the Echoplex is plotted in Figure 12.33.

Tone controls

Tone controls are present on both the Space Echo and Echoplex, and affect the output equalization. Neither unit places the tone controls within the feedback loop. The Echoplex tone circuit is as shown in Figure 12.34. In the figure, 'B' refers to the position of the bass tone control, where $B = 1$ for full boost and $B = 0$ for full cut. The marking 'T' indicates the position of the treble control. Measured frequency responses for various control settings are plotted in the same figure.

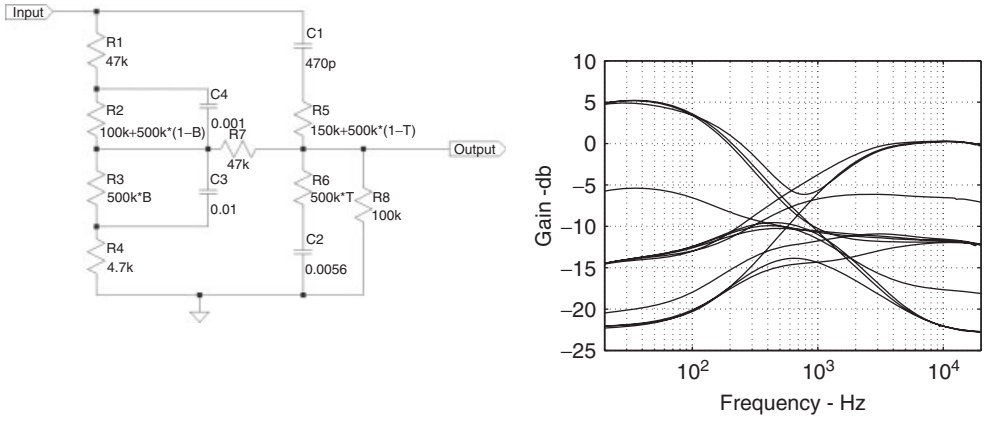

Figure 12.34 Echoplex EP-4 tone circuit and frequency response.

This circuit is related to the Baxandall tone circuit [Bax52] and provides high and low shelving functions. The Baxandall circuit is common in hi-fi and guitar tone stacks. Because of the topology of the circuit, the transfer function cannot be directly computed using series-parallel combinations of the components. However, superposition can be used to decompose the transfer function into two components, each of which can be calculated in a straightforward way. The two superposition components are shown in Figure 12.35.

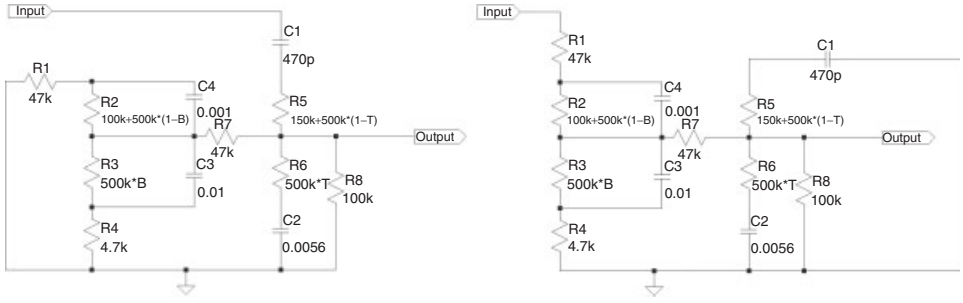

Figure 12.35 Echoplex EP-4 tone circuit superposition components.

The circuit transfer function is identically equal to the sum of the transfer functions of the two superposition components. Because of the topologies of the components, the two transfer functions have the same poles, so that the numerator terms can be added directly. As can be seen, each of the two components can be evaluated directly using series-parallel combinations.

Alternatively, the circuit can be analyzed using the n-extra element theorem (nEET) [Vor02]. Using nEET, the terms of the numerator and denominator of the transfer function can be calculated directly by inspection, using driving-point impedances associated with the reactive components of the circuit.

12.6 Antiquing of audio files

In this section we discuss an algorithm to transform audio files to sound very old. The signal modeling approach is taken here, although the basis of many related algorithms lies in the physical principles of the underlying analog systems [VGKP08].

The antiquing methods introduce simulated degradations to audio signals. They can be divided into two general groups: global and localized degradations [GR98]. Global degradations affect all samples of the audio signal and contain effects such as background noise, wow and flutter, and certain types of nonlinear distortion. Localized degradations affect some signal samples, but not all of them. Examples of such a localized degradation would be the 'clicks' and 'pops' heard on a dusty vinyl record.

Since audio signals are often stereophonic, the first step in audio antiquing is usually to convert the audio signal into the monophonic form. The simplest method, which yields satisfactory results in most cases, is to average the two channels. However, some stereo recordings use special effects, such as time delay or inverted phase between the two channels, and in such cases the simple technique fails. In the worst cases one or more sources in the original signal can be cancelled or be strongly colored, when the channels are added together.

12.6.1 Telephone line effect

The telephone sound is a commonly used effect in music production and in film soundtracks. It is used in movies and TV series to process the actor's voice, when she or he is talking on the phone, but the other person is shown listening. Additionally, it is widely applied to vocals in pop music as a special effect. A straightforward yet fairly realistic audio processing algorithm is given next.

The main component in the telephone sound is its severely limited bandwidth. Even today, telephone systems only transmit signal frequencies between 300 Hz and 3400 Hz. Below and above these cut-off frequencies, the signal energy is greatly suppressed. Figure 12.36 shows the magnitude response of an IIR filter which can be used to model this response. This example filter is a 12th-order Chebyshev type 2 design with cut-off frequencies of 234 Hz and 4300 Hz and a stopband rejection of −40 dB. With these parameters, the −6-dB points are at 300 Hz and 3400 Hz. Applying this filter to a wideband audio signal converts it to a low-bandwidth signal. The sound quality of a musical signal is drastically reduced by this filter. However, there will be no extra disturbances, and voice signals processed like this are perfectly intelligible. Nonetheless, this bandpass filter alone can be used as a simulation of a modern phone.

To obtain a historical telephone sound, it is necessary to add some noise and distortion. For much of the history of the telephone, almost all phone handsets contained a carbon microphone. This is a cheap and robust device, but has problems in sound quality: The output of a carbon microphone is noisy and contains harmonic distortion, which is characterized by a large amount of the second harmonic. This is a consequence of the fact that the resistance of the carbon-button varies in a different manner for positive and negative sound pressures.

The carbon microphone nonlinearity can be modeled with the following asymmetric function [QRO00]:

$$y(n) = \frac{1 - \alpha x(n)}{(1 - \alpha)x(n)}, \tag{12.57}$$

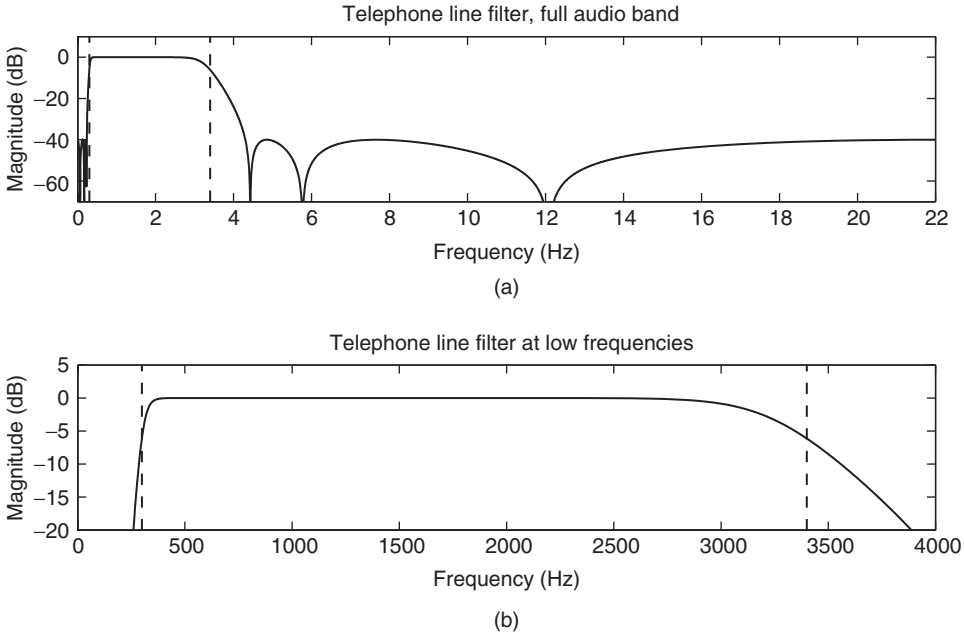

Figure 12.36 Magnitude response of the bandpass filter for telephone sound emulation. The dashed vertical lines indicate the cut-off frequencies 300 Hz and 3400 Hz.

where α determines the amount of nonlinearity when $\alpha > 0$, $x(n)$ is the input signal assumed to vary between -1.0 and 1.0, and $y(n)$ is the output signal. The following simplified nonlinearity can approximately produce the right kind of distortion:

$$y(n) = (1 - \alpha)x(n) + \alpha x^2(n). \tag{12.58}$$

When α is smaller than about 0.4, this nonlinear function can produce a similar asymmetric nonlinear mapping between input and output samples as a physical model of the carbon-button microphone or empirically measured nonlinear functions [QRO00].

To complete the historical telephone sound emulation, a version of the nonlinear sandwich model proposed by Quatieri *et al*. [QRO00] is proposed in the following. The above instantaneous nonlinearity is placed between two linear filters, as shown in Figure 12.37. The role of the first filter, or pre-filter, is to introduce a wide resonance around 2 kHz [QRO00], and the second filter can be the bandpass filter of Figure 12.36. When the sampling rate of the original signal is 44.1 kHz, the following inverse comb filter can be used as the pre-filter:

$$H_{pre}(z) = 0.90 - 0.75z^{-11}, \tag{12.59}$$

which introduces an approximately 4-dB amplification at 2 kHz and at its odd multiples (but the peaks at higher frequencies will be suppressed by the post-filter), and 16-dB dips at dc, at 4 kHz, and at its multiples up to 20 kHz.

White or colored noise may be added to the signal to add realism. In Figure 12.37 it is inserted after the nonlinearity. Although white noise is used here, it will still go through the post-filter and will thus turn into colored, bandpass-filtered hiss. M-file 12.10 gives a compact implementation of the full telephone sound model.

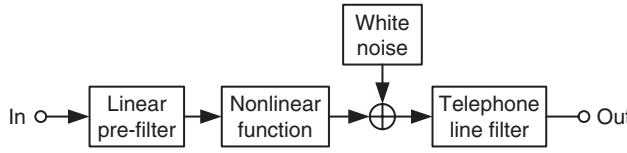

Figure 12.37 Sandwich model for the old telephone sound including the carbon-microphone nonlinearity and a noise source.

M-file 12.10 (phonefx.m)

```
function [y, ylin] = phonefx(alpha, noise, x)
% Authors: Välimäki, Bilbao, Smith, Abel, Pakarinen, Berners
% function [y, ylin] = phonefx(alpha, noise, x)
%
% Parameters:
% alpha = nonlinearity factor (alpha >= 0)
% noise = noise level (e.g. noise = 0.01)
% x = input signal
fs = 44100;  % Sample rate
u = filter([0.9 zeros(1,10) -0.75],1,x);  % Pre-filter
[B,A] = cheby2(6,40,[234/(fs/2) 4300/(fs/2)]);  % Telephone line filter
w = 2*rand(size(x))-1;  % White noise generation
y1 = (1-alpha)*u + alpha*u.^2 ;  % Carbon mic nonlinearity
y2 = y1 + noise*w;    % Add scaled white noise
y = filter(B,A,y2);    % Apply telephone line filter
ylin = filter(B,A,u);  % Linear filtering only (for comparison)
```

12.7 Conclusion

This chapter has focused on virtual analog effects. These digital audio effects are implemented using modern methods, but the aim is to reproduce historical sonorities. In this chapter we have described simulations for various analog filters, tube amplifiers, plate and spring reverberation, tape echo, and an old telephone. Time-domain processing techniques have been applied in all cases. One reason for this is that many algorithms contain nonlinear elements: thus the processing depends on instantaneous sample values and must proceed sample by sample. The nonlinearities, which imitate the behavior of analog components, bring about pleasing distortion and compression, familiar characteristics of nostalgic music equipment.

References

[AAS08] S. Arnardottir, J. S. Abel, and J. O. Smith. A digital model of the Echoplex tape delay. In *Proceedings of the 125th AES Convention*, Preprint 7649, San Francisco, California, October 2008.

[AB79] J. Allen and D. Berkley. Image method for efficiently simulating small-room acoustics. *Journal of Acoustical Society of America*, 65(4): 943–950, 1979.

[ABCS] J. Abel, D. Berners, S. Costello, and J. O. Smith III. Spring reverb emulation using dispersive allpass filters in a waveguide structure. *Presented at the 121st Audio Engineering Society Convention*, San Francisco, California, October, 2006. Preprint 6954.

[Ame03] D. Amels. System and method for distorting a signal, 2003. US Patent No. 6,611,854 B1. Filed September 22, 2000, issued August 26, 2003.

[Arc08] K. Arcas. *Simulation numérique d'un réverbérateur à plaque*. PhD thesis, Ecole Nationale Supérieure de Techniques Avancées, Palaiseau, France, 2008.

[AS96] T. Araya and A. Suyama. Sound effector capable of imparting plural sound effects like distortion and other effects, 1996. US Patent No. 5,570,424. Filed November 24, 1993, issued June 4, 1996.

[BAC06] S. Bilbao, K. Arcas, and A. Chaigne. A physical model of plate reverberation. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pp. 165–168, Toulouse, France, 2006.

[Bax52] P. Baxandall. Negative-feedback tone control. *Wireless World*, pp. 402–405, October 1952.

[Bil01] S. Bilbao. *Wave and Scattering Methods for the Numerical Integration of Partial Differential Equations*. PhD thesis, Stanford University, Palo Alto, CA, USA, 2001.

[Bil07] S. Bilbao. A digital plate reverberation algorithm. *Journal of the Audio Engineering Society*, 55(3): 135–144, 2007.

[Bil09] S. Bilbao. *Numerical Sound Synthesis: Finite Difference Schemes and Simulation in Musical Acoustics*. John Wiley & Sons, Ltd, Chichester, UK, 2009.

[Bot95] D. Botteldooren. Finite-difference time-domain simulation of low-frequency room acoustic problems. *Journal of Acoustical Society of America*, 98(6): 3302–3308, 1995.

[BP10] S. Bilbao and J. Parker. A virtual model of spring reverberation. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(4): 799–808, 2010.

[BPR00] G. Borin, G. De Poli, and D. Rocchesso. Elimination of delay-free loops in discrete- time models of nonlinear acoustic systems. *IEEE Transactions on Speech and Audio Processing*, 8: 597–605, September 2000.

[CL01] A. Chaigne and C. Lambourg. Time-domain simulation of damped impacted plates. I Theory and experiments. *Journal of Acoustical Society of America*, 109(4): 1422–1432, 2001.

[Coo02] R. Cook (ed). *Concepts and Applications of Finite Element Analysis*, 4th edition John Wiley & Sons, Inc., New York, New York, 2002.

[DdV82] L. Della Pietra and S. della Valle. On the dynamic behaviour of axially excited helical springs. *Meccanica*, 17: 31–43, 1982.

[DMRS98] M. Doidic, M. Mecca, M. Ryle, and C. Senffner. Tube modeling programmable digital guitar amplification system, 1998. US Patent No. 5,789,689.

[DST03] G. De Sanctis, A. Sarti, and S. Tubaro. Automatic synthesis strategies for object-based dynamical physical models in musical acoustics. In *Proceedings of the International Conference on Digital Audio Effects*, pp. 219–224, London, England, September 8–11, 2003.

[FC01] P. Fernández-Cid and F. J. Casajús-Quirós. Distortion of musical signals by means of multiband waveshaping. *Journal of New Music Research*, 30(3): 219–287, 2001.

[Fet86] A. Fettweis. Wave digital filters: Theory and practice. *Proceedings of the IEEE*, 74(2): 270–327, February 1986.

[Fon07] F. Fontana. Preserving the structure of the Moog VCF in the digital domain. In *Proceedings of the International Computer Music Conference*, pp. 291–294, Copenhagen, Denmark, August 2007.

[FR91] N. Fletcher and T. Rossing. *The Physics of Musical Instruments*. Springer-Verlag, New York, New York, 1991.

[Gal08] M. N. Gallo. Method and apparatus for distortion of audio signals and emulation of vacuum tube amplifiers, 2008. US Patent Application 2008/0218259 A1. Filed March 6, 2007, published September 11, 2008.

[GCO+04] F. Gustafsson, P. Connman, O. Oberg, N. Odelholm, and M. Enqvist. System and method for simulation of non-linear audio equipment. US Patent Application 20040258250, 2004.

[GR98] S. J. Godsill and P. J. W. Rayner. *Digital Audio Restoration – A Statistical Model Based Approach*. Springer-Verlag, New York, New York, 1998.

[Gra75] K. Graff. *Wave Motion in Elastic Solids*. Dover, New York, New York, 1975.

[Ham41] L. Hammond. Electrical musical instrument, February 2, 1941. US Patent 2,230,836.

[Huo04] A. Huovilainen. Non-linear digital implementation of the Moog ladder filter. In *Proceedings of the International Conference on Digital Audio Effects*, pp. 61–64, Naples, Italy, October 2004.

[Huo05] A. Huovilainen. Enhanced digital models for analog modulation effects. In *Proceedings of the International Conference on Digital Audio Effects*, pp. 155–160, Madrid, Spain, September 20–22, 2005.

[Jac03] D. L. Jackson. Method and apparatus for the modeling and synthesis of harmonic distortion, 2003. US Patent No. 6,504,935 B1. Filed August 19, 1998, issued January 7, 2003.

[JC91] J.-M. Jot and A. Chaigne. Digital delay networks for designing artificial reverberators, 1991. *Presented at the 90th Audio Engineering Society Convention*, Paris, France, February, 1991. Preprint 3030.

[Kar] M. Karjalainen. BlockCompiler documentation. Unfinished report, available on-line at `http:` http://www.acoustics.hut.fi/software/BlockCompiler/docu.html.

[Kar08] M. Karjalainen. Efficient realization of wave digital components for physical modeling and sound synthesis. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(5): 947–956, 2008.

[KI98] R. Kuroki and T. Ito. Digital audio signal processor with harmonics modification, 1998. US Patent No. 5,841,875. Filed January 18, 1996, issued November 24, 1998.

[KMKH06] M. Karjalainen, T. Mäki-Patola, A. Kanerva, and A. Huovilainen. Virtual air guitar. *Journal of the Audio Engineering Society*, 54(10): 964–980, October 2006.

[KP06] M. Karjalainen and J. Pakarinen. Wave digital simulation of a vacuum-tube amplifier. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, Toulouse, France, May 15–19, 2006.

[Kra91] G. Kramer. Digital signal processor for providing timbral change in arbitrary audio and dynamically controlled stored digital audio signals, 1991. US Patent No. 4,991,218 (continuation-in-part of US Pat. 4,868,869). Filed August 24, 1989, issued February 5, 1991.

[Kuh58] W. Kuhl. The acoustical and technological properties of the reverberation plate. *EBU. Review*, A(49), 1958.

[Laa96] T. I. Laakso, V. Välimäki, M. Karjalainen, and U. K. Laine. Splitting the unit delay – Tools for fractional delay filter design. *IEEE Signal Processing Magazine*, 13(1): 30–60, January 1996.

[LCM01] C. Lambourg, A. Chaigne, and D. Matignon. Time-domain simulation of damped impacted plates. II Numerical model and results. *Journal of Acoustical Society of America*, 109(4): 1433–1447, 2001.

[Lea95] W. M. Leach Jr. SPICE models for Vacuum-Tube Amplifiers. *Journal of the Audio Engineering Society*, 43(3): 117–126, 1995.

[LT01] J. Lee and D. Thompson. Dynamic stiffness formulation, free vibration, and wave motion of helical springs. *Journal on Sound and Vibration*, 239(2): 297–320, 2001.

[Mal87] J. Mallinson. *The Foundations of Magnetic Recording*. Academic Press, New York, New York, 1987.

[MGZ02] S. Möller, M. Gromowski, and U. Zölzer. A measurement technique for highly nonlinear transfer functions. In *Proceedings of the International Conference on Digital Audio Effects*, pp. 203–206, Hamburg, Germany, September 26-28, 2002.

[MI68] P. Morse and U. Ingard. *Theoretical Acoustics*. Princeton University Press, Princeton, New Jersey, 1968.

[MKMS07] D. Murphy, A. Kelloniemi, J. Mullen, and S. Shelley. Acoustic modelling using the digital waveguide mesh. *IEEE Signal Processing Magazine*, 24(2): 55–66, 2007.

[Moo65] R. A. Moog. A voltage-controlled low-pass high-pass filter for audio signal processing. In *Proceedings of the 17th Annual Meeting of the Audio Engineering Society*, October 1965.

[MS09] J. Macak and J. Schimmel. Nonlinear circuit simulation using time-variant filter. In *Proceedings of the International Conf. Digital Audio Effects*, Como, Italy, September 1-4, 2009.

[Par08] J. Parker. Spring reverberation: A finite difference approach. *Master's thesis*, University of Edinburgh, 2008.

[PK10] J. Pakarinen and M. Karjalainen. Enhanced wave digital triode model for real-time tube amplifier emulation. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(4): 738–746, 2010.

[PLBC97] E. Pritchard, W. M. Leach Jr, F. Broydé, and E. Clavelier. Comments on "Spice Models for Vacuum-Tube Amplifiers" and Author's Replies. *Journal of the Audio Engineering Society*, 45(6): 488–496, 1997.

[Pri91] E. K. Pritchard. Semiconductor emulation of tube amplifiers, 1991. US Patent No. 4,995,084. Filed March 23, 1998, issued February 19, 1991.

[PTK09] J. Pakarinen, M. Tikander, and M. Karjalainen. Wave digital modeling of the output chain of a vacuum-tube amplifier. In *Proceedings of the 12th International Conference on Digital Audio Effects (DAFx09)*, Como, Italy, September 1–4, 2009.

[Puc82] M. Puckette. Designing multichannel reverberators. *Computer Music Journal*, 6(1): 52–65, 1982.

[PY09] J. Pakarinen and D. T. Yeh. A review of digital techniques for modeling vacuum-tube guitar amplifiers. *Computer Music Journal*, 33(2): 85–100, 2009.

[QPN+07] T. Quarles, D. Pederson, R. Newton, A. Sangiovanni-Vincentelli, and C. Wayne. The spice page, 2007. Internet article http://bwrc.eecs.berkeley.edu/Classes/IcBook/SPICE/ (checked June 5, 2007).

[QRO00] T. F. Quatieri, D. A. Reynolds, and G. C. O'Leary. Estimation of handset nonlinearity with application to speaker recognition. *IEEE Transactions on Speech and Audio Processing*, 8(5): 567–584, September 2000.

[RG75] L. R. Rabiner and B. Gold. *Theory and Application of Digital Signal Processing*. Prentice-Hall, Upper Saddle River, New Jersey, 1975.

[Roc97] D. Rocchesso. Maximally diffusive yet efficient feedback delay networks for artificial reverberation. *IEEE Signal Processing Letters*, 4(9): 252–255, 1997.

[Ros92] D. Rossum. Making digital filters sound "analog". In *Proceedings of the International Computer Music Conference*, pp. 30–33, San Jose, CA, October 1992.

[Ryd95] C. Rydel. Simulation of Electron Tubes with SPICE. In *Proceedings of the 98th AES Convention*, Preprint 3965 (G1), Paris, France, October 25–28, 1995.

[SD99] A. Sarti and G. De Poli. Toward nonlinear wave digital filters. *IEEE Transactions on Signal Processing*, 47(6): 1654–1668, June 1999.

[Ser04] S. Serafin. *The sound of friction: real-time models, playability and musical applications*. PhD thesis, Stanford University, 2004.

[Shi96] M. Shibutani. Distortion circuits for improving distortion effects to audio data, 1996. US Patent No. 5,528,532. Filed June 6, 1995, issued June 18, 1996.

[Sju97] W. Sjursen. Improved SPICE Model for Triode Vacuum Tubes. *Journal of the Audio Engineering Society*, 45(12): 1082–1088, 1997.

[SM07] J. Schimmel and J. Misurec. Characteristics of broken-line approximation and its use in distortion audio effects. In *Proceedings of the International Conference on Digital Audio Effects*, pp. 161–164, Bordeaux, France, September 10–15, 2007.

[Smi84] J. O. Smith. An all-pass approach to digital phasing and flanging. In *Proceedings of the International Computer Music Conference*, pp. 103–109, 1984.

[Smi07] J. O. Smith III. *Introduction to Digital Filters with Audio Applications*. `http://ccrma.stanford.edu/~jos/filters/`, September 2007. Online book.

[Smi08] J. O. Smith III. Virtual electric guitars and effects using Faust and Octave. In *Proceedings of the 6th Int. Linux Audio Conf. (LAC2008)*, 2008. paper and supporting website: `http://ccrma.stanford.edu/realsimple/faust_strings/`, presentation overheads: `http://ccrma-ftp.stanford.edu/~jos/pdf/LAC2008-jos.pdf`.

[SS96] T. Stilson and J. Smith. Analyzing the Moog VCF with considerations for digital implementation. In *Proceedings of the International Computer Music Conference*, pp. 398–401, Hong Kong, China, August 1996.

[Smi10] J. O. Smith. *Physical Audio Signal Processing: for Virtual Musical Instruments and Digital Audio Effects.*. 2010. On-line version at `https://ccrma.stanford.edu/~jos/pasp/`.

[SST07] F. Santagata, A. Sarti, and S. Tubaro. Non-linear digital implementation of a parametric analog tube ground cathode amplifier. In *Proceedings of the International Conference on Digital Audio Effects*, pp. 169–172, Bordeaux, France, September 10–15, 2007.

[Sta48] J. D. Stack. Sound reverberating device, March 9 1948. US Patent 2,437,445.

[Ste96] K. Steiglitz. *A Digital Signal Processing Primer with Applications to Digital Audio and Computer Music*. Addison-Wesley, Reading, Massachusetts, 1996.

[Sti06] T. Stilson. *Efficiently Variable Non-Oversampled Algorithms in Virtual-Analog Music Synthesis – A Root-Locus Perspective*. PhD thesis, Elec. Eng. Dept., Stanford University (CCRMA), June 2006. `http://ccrma.stanford.edu/~stilti/`.

[Str89] J. Strikwerda. *Finite Difference Schemes and Partial Differential Equations*. Wadsworth and Brooks/Cole Advanced Books and Software, Pacific Grove, California, 1989.

[Sul90] C. S. Sullivan. Extending the Karplus-Strong algorithm to synthesize electric guitar timbres with distortion and feedback. *Computer Music Journal*, 14(3): 26–37, fall 1990.

[Szi74] R. Szilard. *Theory and Analysis of Plates*. Prentice Hall, Englewood Cliffs, New Jersey, 1974.

[VGKP08] V. Välimäki, S. González, O. Kimmelma, and J. Parviainen. Digital audio antiquing – signal processing methods for imitating the sound quality of historical recordings. *Journal of the Audio Engineering Society*, 56(3): 115–139, March 2008.

[VH06] V. Välimäki and A. Huovilainen. Oscillator and filter algorithms for virtual analog synthesis. *Computer Music Journal*, 30(2): 19–31, 2006.

[Vor02] V. Vorpérian. *Fast Analytical Techniques for Electrical and Electronic Circuits*. Cambridge University Press, 2002.

[VPA10] V. Välimäki, J. Parker, and J. S. Abel. Parametric spring reverberation effect. *Journal of the Audio Engineering Society*, 58(7/8): 547–562, July/August 2010.

[VPEK06] V. Välimäki, J. Pakarinen, C. Erkut, and M. Karjalainen. Discrete-time modelling of musical instruments. *Reports on Progress in Physics*, 69(1): 1–78, January 2006.

[Wis98] D. K. Wise. The recursive allpass as a resonance filter. In *Proceedings of the International Computer Music Conference*, Ann Arbor, MI, October 1998.

[Wit66] W. Wittrick. On elastic wave propagation in helical springs. *International Journal of Mechanical Sciences*, 8: 25–47, 1966.

[YAAS08] D. T. Yeh, J. S. Abel, A. Vladimirescu, and J. O. Smith. Numerical methods for simulation of guitar distortion circuits. *Computer Music Journal*, 32(2): 23–42, 2008.

[Yeh09] D. T. Yeh. *Digital Implementation of Musical Distortion Circuits by Analysis and Simulation*. PhD thesis, CCRMA, Elec. Eng. Dept., Stanford University (CCRMA), Palo Alto, CA, USA, June 2009. http:http://ccrma.stanford.edu/dtyeh//ccrma.stanford.edu/~dtyeh/.

[YI63] A. C. Young and P. It. Artificial reverberation unit, October 8, 1963. US Patent 3,106,610.

[YS06] D. Yeh and J. O. Smith III. Discretization of the 59 Fender Bassman tone stack. In *Proceedings of the 9th International Conference on Digital Audio Effects (DAFx06)*, September 2006. http://www.dafx.de/.

[YS08] D. T. Yeh and J. O. Smith. Simulating guitar distortion circuits using wave digital and nonlinear state-space formulations. In *Proceedings of the International Conference on Digital Audio Effects*, pp. 19–26, Espoo, Finland, September 1–4, 2008.