Julius O. Smith III
Center for Computer Research in Music and Acoustics (CCRMA)
Department of Music, Stanford University
Stanford, California 94305 USA
jos@CCRMA.Stanford.edu

Physical Modeling Using Digital Waveguides

Music synthesis based on a physical model promises the highest quality when it comes to imitating natural instruments. Because the artificial instrument can have the same control parameters as the real instrument, expressivity of control is unbounded.

Historically, physical models have led to prohibitively expensive synthesis algorithms, and commercially available synthesizers do not yet appear to make use of them. These days, most synthesizers use either processed digital recordings ("sampling synthesis") or an abstract algorithm such as frequency modulation (FM). However, as computers become faster and cheaper, and as algorithms based on physical models become more efficient, we may expect to hear more from them.

Most attempts to synthesize sounds based on a physical model have been based on numerical integration of the wave equation (covered in any textbook on acoustics). These methods generally require at least one operation (multiplication and/or addition) for each point on a grid that permeates the instrument. In principle, the grid spacing must be less than half a wavelength at the highest audio frequency. This is essentially why the computational costs are so high in "brute force" numerical solutions of the wave equation.

More recently developed "digital waveguide" methods follow a different path to the physical model; the wave equation is first solved in a general way to obtain traveling waves in the medium interior. The traveling waves are explicitly simulated in the waveguide model, in contrast to computing a physical variable. (The traveling waves must be summed together to produce a physical output.) In the lossless case, a traveling wave between two points in the medium can be simulated using nothing but a digital delay line. In the general linear case, in which there are frequency-dependent losses and dispersion, the commutativity of linear time-invariant systems allows the losses and dispersion to be lumped at discrete points such that most of the simulation still consists of multiply-free delay lines. This is essentially why computational costs are so low in "waveguide synthesis" algorithms.

Computer-music programmers know very well that a delay line can be implemented by a single fetch, store, and pointer update for each sample of output. If the delay line is, say, 500 samples long (corresponding to a pitch of 44,100/500=88 Hz in a string or bore model of compact disk quality), computational requirements relative to "brute force" numerical integration on the grid are reduced by three orders of magnitude. As a result, for very simple physical models, several CD-quality voices can be sustained in real time on a single digital signal processing (DSP) chip costing only a few dollars.

This article develops waveguide synthesis beginning with the wave equation for vibrating strings. Transverse waves on a string are taken as the primary example due to the relative clarity of the underlying physics, but the formulation for string simulation is unified with that of the acoustic tube. The technique of lumping losses at discrete points in the waveguide, replacing more expensive distributed losses, is described.

Computer Music Journal, Vol. 16, No. 4, Winter 1992,

The Ideal Vibrating String

The wave equation for the ideal (lossless, linear, flexible) vibrating string, depicted in Fig. 1 , is given by

$$Ky'' = \varepsilon\ddot{y},$$

where

$K \triangleq$ string tension

$\varepsilon \triangleq$ linear mass density

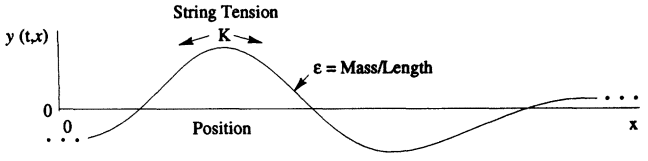
$y \triangleq$ string displacement

$y \triangleq y(t,x)$

$\dot{y} \triangleq \frac{\partial}{\partial t} y(t,x)$

$y' \triangleq \frac{\partial}{\partial x} y(t,x)$

Fig.1. The ideal vibrating string.



The wave equation was fully derived by Morse (1936) and appears in most elementary textbooks on acoustics. It can be interpreted as a statement of Newton's law, *"force = mass × distance,"* on a microscopic scale. Since we are concerned with transverse vibrations on the string, the relevant restoring force (per unit length) is given by the string tension times the curvature of the string (Ky''); the restoring force is balanced at all times by the inertial force per unit length of the string, which is equal to mass density times transverse acceleration ($e\ddot{y}$).

The same wave equation applies to any perfectly elastic medium that is displaced along one dimension. For example, the air column of a clarinet or organ pipe can be modeled using the one-dimensional wave equation, substituting air pressure deviation for string displacement, and longitudinal volume velocity of air in the bore for transverse velocity on the string. We refer to the general class of such media as *one-dimensional waveguides*. Extensions to two and three dimensions (and more, for the mathematically curious) are also possible.

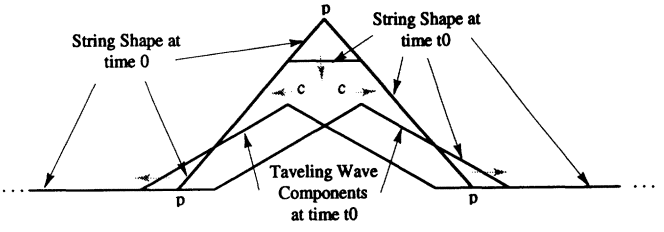
Traveling-Wave Solution

It can be readily checked that the wave equation is solved by any string shape which travels to the left or right with speed $c = \sqrt{K/\varepsilon}$. If we denote rightgoing traveling waves by $y_r(x - ct)$ and leftgoing traveling waves by $y_\ell = (x + ct)$, where y_r and y_ℓ are arbitrary twice-differentiable functions, then the general class of solutions to the lossless, one-dimensional, second-order wave equation can be expressed as

$$y(x,t) = y_r(x - ct) + y_\ell(x + ct).$$

Note that $\ddot{y}_r = c^2 y''_r$ and $\ddot{y}_\ell = c^2 y''_\ell$, which shows that the wave equation is obeyed regardless of the traveling wave shapes y_r and y_ℓ. (But note that the derivation of the wave equation itself assumes that the string slope is much less than 1 at all times and positions.) The traveling-wave solution of the wave equation was first published by d'Alembert in 1747.

Fig. 2. An infinitely long string, "plucked" simultaneously at three points (labeled p*), so as to produce an initial triangular displacement. The initial displacement is modeled as the sum of two identical triangular pulses which are exactly on top of each other at time 0. At time* t_0 *shortly after time 0, the traveling waves have begun to separate, and their sum gives the physical string displacement at time* t_0*, which is also shown. Note that only three short string segments are in motion at that time: the flat top segment, which is heading to zero where it will halt forever, and two short pieces on the left and right, which are the leading edges of the leftgoing and rightgoing traveling waves. The portion of the string where the traveling waves overlap is not moving. When the traveling waves fully separate, the string will be at rest everywhere but for two half-amplitude triangular pulses heading off to plus and minus infinity at speed* C.



An example of the appearance of the traveling wave components shortly after plucking an infinitely long string at three points is shown in Fig. 2.

Sampling the Traveling Waves

To carry the traveling-wave solution into the "digital domain," it is necessary to sample the traveling-wave amplitudes at intervals of T seconds, corresponding to a sampling rate $f_s \doteq 1/T$ samples per second. This leads to the basic digital waveguide structure for simulating uniform propagation media (Smith 1985b, 1987, 1990). For CD-quality audio, we have $f_s = 44.1$ kHz. The natural choice of spatial sampling interval X is the distance sound propagates in one temporal sampling interval T, or $X \doteq cT$ meters. In air, assuming the speed of sound to be 331 m/sec, we have $X = 331/44{,}100 = 7.5$ mm for the spatial sampling interval, or a spatial sampling rate of 133 samples per meter. In a traveling-wave simulation, the whole wave moves left or right by one spatial sample each time sample; hence, simulation only requires digital delay lines.

Fig. 3. Digital simulation of the ideal, lossless waveguide with observation points at X = 0 and X = 3X = 3cT. The symbol "Z^{-1}" denotes a one-sample delay.

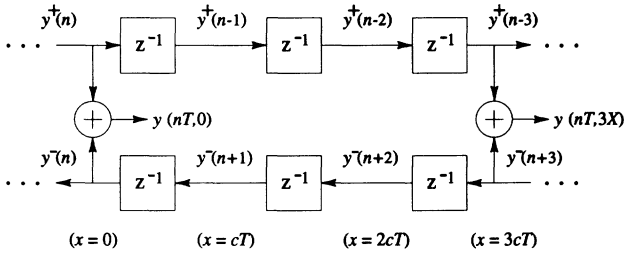
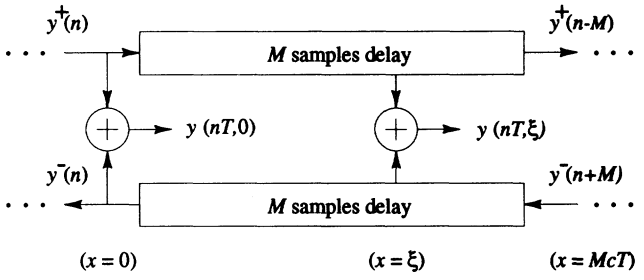


Fig. 4. Simplified picture of ideal waveguide simulation.



Formally, sampling is carried out by the change of variables

$$x \rightarrow x_m = mX,$$
$$t \rightarrow t_n = nT.$$

Substituting into the traveling-wave solution of the wave equation gives

$$\begin{aligned} y(t_n, x_m) &= y_r(t_n - x_m/c) + y_\ell(t_n + x_m/c) \\ &= y_r(nT - mX/c) + y_\ell(nT + mX/c) \\ &= y_r[(n-m)T] + y_\ell[(n+m)T]. \end{aligned}$$

Since T multiplies all arguments, let's suppress it by defining

$$y^+(n) \triangleq y_r(nT), \quad y^-(n) \triangleq y_\ell(nT).$$

This new notation also introduces a "plus" (+) superscript to denote a traveling-wave component propagating to the right, and a "minus" (–) superscript to denote propagation to the left.

The term $y_r[(n-m)T] = y^+(n-m)$ can be thought of as the output of an m-sample delay line whose input is $y^+(n)$. In general, subtracting a positive number m from a time argument n corresponds to delaying the waveform by m samples. Since y^+ is the rightgoing component, we draw its delay line with input $y^+(n)$ on the left and its output $y^+(n-m)$ on the right. This can be seen as the upper "rail" in Fig. 3.

Similarly, the term $y_\ell[(n+m)T] \doteq y^-(n+m)$ can be thought of as the input to an m-sample delay line whose output is $y^-(n)$. (Adding m to the time argument n produces an m-sample waveform advance.) Since y^- is the leftgoing component, it makes sense to draw the delay line with its input $y^-(n+m)$ on the right and its output $y^-(n)$ on the left. This can be seen as the lower "rail" in Fig. 3. Note that the position along the string, $x_m = mX = mcT$ meters, is laid out from left to right in the diagram, giving a physical interpretation to the horizontal direction in the diagram. Finally, the leftgoing and rightgoing traveling waves must be summed to produce a physical output according to the formula

$$y(t_n, x_m) = y^+(n-m) + y^-(n+m).$$

We may compute the physical string displacement at any spatial sampling point x_m by simply adding the upper and lower rails together at position m along the delay-line pair. In Fig. 3, "transverse displacement outputs" have been arbitrarily placed at $x = 0$ and $x = 3X$.

The Appendix contains a C program that illustrates explicitly the implementation of a diagram such as the above in the context of a plucked string simulation. A more compact simulation diagram, which stands for either sampled or continuous simulation is shown in Fig. 4. Figure 4 emphasizes that the ideal, lossless waveguide is simulated by a bidirectional delay line. Any ideal one-dimensional waveguide can be simulated in this way. It is important to note that the simulation is exact at the sampling instants, to within the numerical precision of the samples themselves, provided the waveshapes traveling along the string are initially bandlimited to less than half the sampling frequency. In other words, the highest frequencies present in the signals $y_r(t)$ and $y_\ell(t)$ may not exceed half the temporal sampling frequency $f_s \doteq 1/T$; equivalently, the highest spatial frequencies in the shapes $y_r(x/c)$ and

Fig. 5. Conversions between various time derivatives of displacement: y = *displacement,* v = $\dot{y}$ = *velocity,* a = $\ddot{y}$ = *acceleration.*

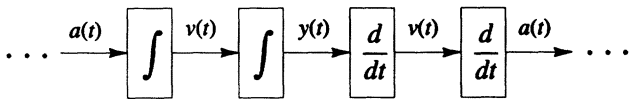
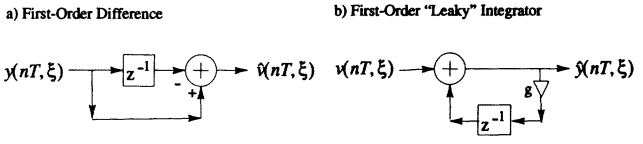


Fig. 6. Simple approximate conversions between time derivatives in the discrete-time case. (a) The first-order difference. (b) The first-order "leaky" integrator with loss factor g *(slightly less than 1) used to avoid infinite direct current buildup.*



$y_\ell(x/c)$ may not exceed half the spatial sampling frequency $v_s \doteq 1/X$.

Bandlimited spatial interpolation may be used to construct a displacement output for an arbitrary x not a multiple of cT, as suggested by the output drawn in Fig. 4. Similarly, bandlimited interpolation across time serves to evaluate the waveform at an arbitrary time not an integer multiple of T.

Ideally, bandlimited interpolation is carried out by convolving a continuous "sinc function" $\text{sinc}(x) \doteq \sin(\pi x)/\pi x$ with the signal samples. Specifically, convolving sampled signal $x(t_n)$ with $\text{sinc}[(t_n - t_0)/T]$ "evaluates" the signal at an arbitrary continuous time t_0. The sinc function is the impulse response of the ideal lowpass filter that cuts off at half the sampling rate.

In practice, the interpolating sinc function must be *windowed* to a finite duration. This means the associated lowpass filter must be granted a "transition band" in which its frequency response is allowed to "roll off" to zero at half the sampling rate. The interpolation quality in the "pass band" can always be made perfect to within the resolution of human hearing by choosing a sufficiently large product of window-length times transition-bandwidth. Given "audibly perfect" quality in the pass band, increasing the transition bandwidth reduces the computational expense of the interpolation. This is one reason why oversampling at rates higher than twice the highest audio frequency is helpful. This topic is described further elsewhere (Smith and Gossett 1984).

Alternative Wave Variables

We have thus far considered discrete-time simulation of traveling displacement waves $y^{\pm}$ in the ideal string. It is equally valid to choose traveling velocity $v^{\pm} \doteq y^{\pm}$, acceleration $a^{\pm} \doteq \ddot{y}^{\pm}$, or slope waves $y'^{\pm}$, or perhaps some other derivative or integral of displacement with respect to time or position or both.

Conversion between various time derivatives can be carried out by means of integrators or differentiators, as depicted in Fig. 5. Since integration and differentiation are linear operators, and since the traveling-wave arguments are in units of time, the same conversion formulas hold for the traveling-wave components $y^{\pm}$,$v^{\pm}$, and $a^{\pm}$.

In discrete time, integration and differentiation can be accomplished using digital filters (Rabiner and Gold 1975). Commonly used approximations are shown in Fig. 6. These first-order approximations are accurate (though scaled by T) at low frequencies relative to half the sampling rate, but they are not "best" approximations in any sense other than being most like the definitions of integration and differentiation in continuous time. Much better approximations can be obtained by approaching the problem from a digital filter design viewpoint (Rabiner and Gold 1975; Smith 1985a; Parks and Burrus 1987; Loy 1988). Arbitrarily better approximations are possible using higher order digital filters. In principle, a digital differentiator is a filter whose frequency response $H(e^{j\omega T})$ optimally approximates jω for ω between $-\pi T$ and πT. Similarly, a digital integrator must match $1/j\omega$ along the unit circle in the z plane. An exact match is not possible because the ideal frequency responses $j\omega$ and $1/j\omega$, when wrapped along the unit circle in the z plane (which is the frequency axis for discrete time systems), are not "simple" functions any more. As a result, there is no filter with a rational transfer function (i.e., finite order) that can match the desired frequency response exactly.

Spatial Derivatives

In addition to time derivatives, we may apply any number of spatial derivatives to obtain yet more wave variables to choose from. The first spatial derivative of string displacement yields slope waves:

$$y'(t,x) \triangleq \frac{\partial}{\partial x} y(t,x) = y'_r(t - x/c) + y'_\ell(t + x/c)$$
$$= -\frac{1}{c}\dot{y}_r(t - x/c) + \frac{1}{c}\dot{y}_\ell(t + x/c),$$

or, in discrete time,

$$y'(t_n, x_m) \triangleq y'(nT, mX) = y'_r[(n - m)T]$$
$$+ y'_\ell[(n + m)T] \triangleq y'^{+}(n - m) + y'^{-}(n + m)$$
$$= -\frac{1}{c}\dot{y}^{+}(n - m) + \frac{1}{c}\dot{y}^{-}(n + m) \triangleq -\frac{1}{c}v^{+}(n - m)$$
$$+ \frac{1}{c}v^{-}(n + m)$$
$$= \frac{1}{c}[v^{-}(n + m) - v^{+}(n - m)].$$

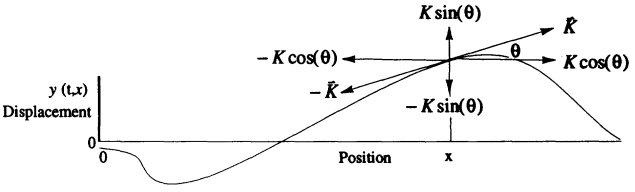
From this we conclude that $v^- = cy'^-$ and $v^+ = -cy'^+$. That is, traveling slope waves can be computed from traveling velocity waves by dividing by c and negating in the rightgoing case. Physical string slope can thus be computed from a velocity-wave simulation by subtracting the upper rail from the lower rail and dividing by c.

By the wave equation, curvature waves, $y'' = \ddot{y}/c^2$, are essentially identical to acceleration waves.

In the field of acoustics, the state of a vibrating string at any instant of time t_0 is normally specified by the displacement $y(t_0,x)$ and velocity $\dot{y}(t_0,x)$ for all x. Since displacement is the sum of the traveling displacement waves and velocity is proportional to the difference of the traveling displacement waves, one state description can be readily obtained from the other.

In summary, all traveling-wave variables can be computed from any one, as long as both the leftgoing and rightgoing component waves are available. Alternatively, any two linearly independent physical variables, such as displacement and velocity, can be used to compute all other wave variables. Wave variable conversions requiring differentiation or integration are relatively expensive since a large-order digital filter is necessary to do them correctly. Slope waves can be computed from velocity waves by a simple scaling, and vice versa, and curvature waves are the same as acceleration waves to within a scale factor.

Fig. 7. Transverse force propagation in the ideal string.



In the absence of other factors dictating a choice, velocity waves are a good overall choice because (1) it is numerically easier to perform digital integration to get displacement than it is to differentiate displacement to get velocity, and (2) slope waves are immediately computable from velocity waves. Why slope waves are important is the subject of the next section.

Force Waves

Referring to Fig. 7, at an arbitrary point x along the string, the vertical force applied at time t *to* the portion of string to the left of position x *by* the portion of string to the right of position x is given by

$$f_\ell(t,x) = K\sin(\theta) \approx K\tan(\theta) = Ky'(t,x),$$

assuming $|y'(t,x)| << 1$, as is assumed in the derivation of the wave equation. Similarly, the force applied *by* the portion to the left of position x *to* the portion to the right is given by

$$f_r(t,x) = -K\sin(\theta) \approx -Ky'(t,x).$$

These forces must cancel since a nonzero net force on a massless point would produce infinite acceleration.

Vertical force waves propagate along the string like any other transverse wave variable (since they are just slope waves multiplied by tension K). We may choose either f_ℓ or f_r as the string force wave variable, one being the negative of the other. It turns out that to obtain a unification of vibrating strings and air columns, we have to pick f_r, the one that acts to the right. This makes sense intuitively when one considers longitudinal pressure waves in an acoustic

tube: a compression wave traveling to the right in the tube pushes the air in front of it and thus acts to the right. Thus, we define the force wave variable to be

$$f(t,x) \triangleq f_r(t,c) = -Ky'(t,x).$$

Substituting from above, we have

$$f(t,x) = \frac{K}{c}[\dot{y}_r(t - x/c) - \dot{y}_\ell(t + x/c)].$$

Note that $K/c \triangleq K/\sqrt{K/\varepsilon} = \sqrt{K\varepsilon}$. This is a fundamental quantity known as the *wave impedance* of the string (also called the *characteristic impedance*), denoted as

$$R \triangleq \sqrt{K\varepsilon}.$$

The wave impedance can be seen as the geometric mean of the two resistances to displacement: tension (spring force) and mass (inertial force). Note that $R = K/c = \varepsilon c$ since $c = \sqrt{K/\varepsilon}$.

The digitized, traveling, force-wave components become

$$f^+(n) = Rv^+(n),$$

$$f^-(n) = -Rv^-(n),$$

which gives us that the rightgoing force wave equals the wave impedance times the rightgoing velocity wave, and the leftgoing force wave equals *minus* the wave impedance times the leftgoing velocity wave. Thus, in a traveling wave, force is always *in phase* with velocity (considering the minus sign in the leftgoing case to be associated with the direction of travel rather than a 180° phase shift between force and velocity). Note that if the leftgoing force wave were defined as the string force acting to the left, the minus sign would disappear.

In the case of the acoustic tube (Morse 1936; Markel and Gray 1976), we have the analogous relations

$$p^+(n) = Ru^+(n),$$

$$p^-(n) = -Ru^-(n),$$

where $p^+(n)$ is the rightgoing traveling longitudinal pressure wave, $p^-(n)$ is the leftgoing pressure wave, and $u^\pm(n)$ are the leftgoing and rightgoing volume velocity waves. In the acoustic tube context, the wave impedance is given by

$$R = \frac{\rho c}{A},$$

where ρ is the mass per unit volume of air, c is sound speed, and A is the cross-sectional area of the tube. Note that if we had chosen particle velocity, rather than volume velocity, the wave impedance would have been $R = \rho c$ instead, the wave impedance in open air.

Power Waves

Basic courses in physics teach us that *power* is *work per unit time*, and *work* is a measure of energy, which is typically defined as *force times distance*. Therefore, power is in physical units of force times distance per unit time, or force times velocity. It therefore should come as no surprise that traveling power waves are defined as

$$P^+(n) \triangleq f^+(n)v^+(n),$$
$$P^-(n) \triangleq -f^-(n)v^-(n).$$

Note that $P^+(n) = f^+(n)v^+(n) = R[v^+(n)]^2$, and $R, (n) = -f^-(n)v^-(n) = R[v^-(n)]^2 = [f^-(n)]^2/R$. Thus both the leftgoing and rightgoing components are nonnegative. The sum of the traveling powers at a point thus gives the total power at that point on the string.

$$P(t_n, x_m) \triangleq P^+(n - m) + P^-(n + m).$$

If we had left out the minus sign in the definition of leftgoing power waves, the sum would instead be a net power flow.

Power waves are important for several reasons. They correspond to the actual ability of the wave to do work on the outside world, such as on a violin bridge at the end of the string. Because energy is conserved in closed systems, power waves sometimes give a simpler, more fundamental view of wave phenomena, as in the case of conical acoustic tubes.

Fig. 8. Discrete simulation of the ideal, lossy waveguide. The loss factor $g = e^{-\mu T \varepsilon}$ summarizes the distributed loss incurred in one sampling period.

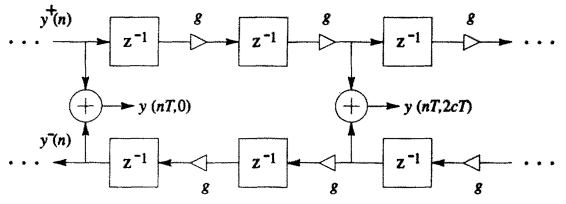
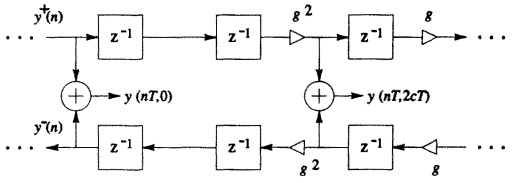


Fig. 9. Discrete simulation of the ideal, lossy waveguide. Each per-sample loss factor g is "pushed through" delay elements and combined with other loss factors until an input or output is encountered which inhibits further migration. If further consolidation is possible on the other side of a branching node, a loss factor can be pushed through the node by pushing a copy into each departing branch. If there are other inputs to the node, the inverse of the loss factor must appear on each of them. Similar remarks apply to pushing backward through a node.



Energy Density Waves

The vibrational energy per unit length along the string, or *wave energy density* (Morse 1936), is given by the sum of potential and kinetic energy densities

$$W(t,x) \triangleq \frac{1}{2}Ky'^2(t,x) + \frac{1}{2}\varepsilon\dot{y}^2(t,x).$$

Sampling across time and space, and substituting traveling wave components, one can show in a few lines of algebra that the sampled wave energy density is given by

$$W(t_n,x_m) \triangleq W^+(n-m) + W^-(n+m),$$

where

$$W^+(n) \triangleq \frac{P^+(n)}{c} \triangleq \frac{f^+(n)v^+(n)}{c} = \varepsilon[v^+(n)]^2 = \frac{[f^+(n)]^2}{K},$$
$$W^-(n) \triangleq \frac{P^-(n)}{c} \triangleq \frac{f^-(n)v^-(n)}{c} = \varepsilon[v^-(n)]^2 = \frac{[f^-(n)]^2}{K}.$$

Thus, traveling power waves (energy per unit time) can be converted to energy density waves (energy per unit length) by simply dividing by c, the speed of propagation. Quite naturally, the total wave energy in the string is given by the integral along the string of the energy density

$$\varepsilon(t) \triangleq \int_{x=-\infty}^{\infty} W(t,x)dx \approx \sum_{m=-\infty}^{\infty} W(t,x_m)X.$$

In practice, of course, the string length is finite, and the limits of integration/summation are from the x coordinate of the left endpoint to that of the right endpoint (e.g., 0 to L).

The Lossy One-Dimensional Wave Equation

In any real vibrating string, there are energy losses due to yielding terminations, drag by the surrounding air, and internal friction within the string. Many kinds of loss exert a resistive force proportional to transverse velocity of the string over a range of forces. If this proportionality constant is μ, we obtain the modified wave equation

$$Ky'' = \varepsilon\ddot{y} + \mu\dot{y}.$$

It can be checked that for small displacements, the following modified traveling-wave solution satisfies the lossy wave equation:

$$y(t,x) = e^{-(\mu/2\varepsilon)x/c}y_r(t-x/c) + e^{(\mu/2\varepsilon)x/c}y_\ell(t+x/c).$$

The leftgoing and rightgoing traveling-wave components decay exponentially in their respective directions of travel. Sampling these exponentially decaying traveling waves at intervals of T seconds (or $X = cT$ meters) gives

$$y(t_n,x_m) = g^{-m}y^+(n-m) + g^m y^-(n+m),$$

where $g \triangleq e^{-\mu T/2\varepsilon}$.

The digital simulation diagram for the lossy waveguide is shown in Fig. 8. Again note that the simulation of the decaying traveling-wave solution is exact at the sampling positions and instants, even though losses are admitted in the wave equation. Note also that the losses which are distributed in the continuous solution have been consolidated, or

lumped, at discrete intervals of cT meters in the simulation. The lumping of distributed losses does not introduce an approximation error at the sampling points. Furthermore, bandlimited interpolation can yield arbitrarily accurate reconstruction between samples. The only restriction is that all initial conditions and excitations be bandlimited to half the sampling rate.

Consolidating Losses on a Larger Scale

In many applications, it is possible to realize vast computational savings in waveguide simulation by commuting losses out of unobserved and undriven sections of the medium and consolidating them at a minimum number of points. Because the digital simulation is linear and time invariant (given constant medium parameters K, ε, and μ), and because linear, time-invariant elements commute, the diagram in Fig. 9 is exactly equivalent (to within numerical precision) to the previous diagram in Fig. 8.

Frequency-Dependent Losses

In nearly all natural wave phenomena, losses increase with frequency. The largest losses in a real stringed instrument occur at the bridge, particularly at frequencies that couple with body resonances. There are also losses due to air drag and internal bulk losses in the string, which increase monotonically with frequency. Similarly, air absorption increases with frequency, providing an increase in propagation loss for sound waves in air (Morse and Ingard 1968).

The solution to the lossy wave equation can be generalized to the frequency-dependent case. Instead of factors g distributed throughout the diagram, the factors are now $G(\omega)$. These loss factors, implemented as digital filters, may also be consolidated at a minimum number of points in the waveguide without introducing an approximation error.

In an efficient digital simulation, each lumped loss factor $G(\omega)$ is to be approximated by a rational frequency response $\hat{G}(e^{j\omega T})$. In general, the coefficients of the optimal rational loss filter are obtained by minimizing $\|G(\omega) - \hat{G}(e^{j\omega t})\|$ with respect to the filter coefficients or the poles and zeros of the filter. To avoid introducing frequency-dependent delay, the loss filter should be a zero-phase, finite-impulse-response (FIR) filter (Rabiner and Gold 1975). Restriction to zero phase requires the impulse response $\hat{g}(n)$ to be finite in length (i.e., an FIR filter), and it must be symmetric about time zero, that is, $\hat{g}(-n) = \hat{g}(n)$. For real-time implementations, the zero-phase FIR filter can be converted into a causal linear phase filter by "stealing" delay from the loop delay lines in an amount equal to half the impulse response duration.

The Dispersive One-Dimensional Wave Equation

Stiffness in a vibrating string introduces a restoring force proportional to the fourth derivative of the string displacement (Morse 1936; Cremer 1984):

$$\varepsilon\ddot{y} = Ky'' - \kappa y'''',$$

where, for a cylindrical string of radius a and Young's modulus Q, the moment constant κ is equal to $\kappa = Q\pi a^4/4$.

At very low frequencies, or for very small κ, we return to the nonstiff case. At very high frequencies, or for very large κ, we approach the ideal bar in which stiffness is the only restoring force. At intermediate frequencies, between the ideal string and bar, the stiffness contribution can be treated as a correction term (Cremer 1984). This is the region of most practical interest because it is the principal operating region for strings, such as piano strings, whose stiffness has audible consequences (an inharmonic, stretched overtone series). The first-order effect of stiffness is to increase the wave propagation speed with frequency:

$$c(\omega) \triangleq c_0\left(1+\frac{\kappa\omega^2}{2Kc_0^2}\right),$$

where c_0 is the wave travel speed in the absence of stiffness. Since sound speed depends on frequency, traveling waveshapes will "disperse" as they propagate along the string. That is, a traveling wave is no

Fig. 10. The rigidly terminated ideal string, with a position output indicated at X = ξ. Rigid terminations reflect traveling displacement, velocity, or acceleration waves with a sign inversion. Slope or force waves reflect with no sign inversion.

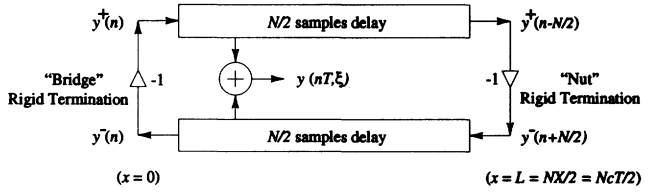


Fig. 11. A doubly terminated string, "plucked" at one-fourth its length.

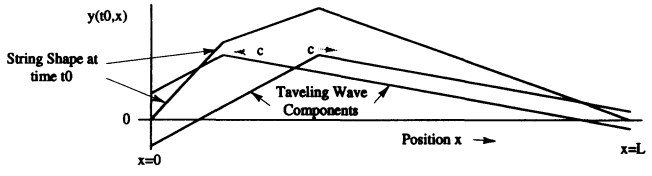
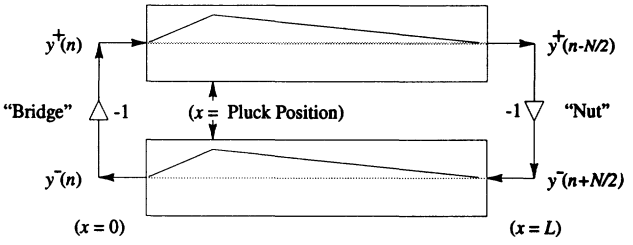


Fig. 12. Initial conditions for the ideal plucked string. The initial contents of the sampled, traveling-wave delay lines are in effect plotted inside the delay-line boxes. The amplitude of each traveling-wave delay line is half the amplitude of the initial string displacement. The sum of the upper and lower delay lines gives the actual initial string displacement.



longer a static shape moving with speed c and expressible as a function of $t \pm x/c$. In a stiff string, the high frequencies propagate faster than the low-frequency components. As a result, a traveling velocity step, such as would be caused be a hammer strike, "unravels" into a smoother velocity step with high-frequency "ripples" running out ahead.

In a digital simulation, a frequency-dependent speed of propagation can be implemented using allpass filters which have a nonuniform delay versus frequency. Note that every linear, time-invariant filter can be expressed as a zero-phase filter in cascade with an allpass filter. The zero-phase part implements frequency-dependent gain (damping in a digital waveguide), and the allpass part gives frequency-dependent delay, which in a digital waveguide yields dispersion. Every linear wave equation with constant coefficients, regardless of its order, corresponds to a waveguide that can be modeled as a pure delay and a linear, time-invariant filter that simulate propagation over a given distance.

Rigid Terminations

A rigid termination is the simplest case of a string termination. It imposes the constraint that the string cannot move at all at the termination. If we terminate a length L ideal string at $x = 0$ and $x = L$, we then have the "boundary conditions"

$$y(t,0) \equiv 0 \qquad y(t,L) \equiv 0$$

for all t. Since $y(t,0) = y_r(t) + y_\ell(t) = y^+(t/T) + y^-(t/T)$ and $y(t,L) = y_r(t - L/c) + y_\ell(t + L/c)$, the constraints on the sampled traveling waves become

$$y^+(n) = -y^-(n),$$

$$y^-(n + N/2) = -y^+(n - N/2),$$

where $N \triangleq 2L/X$ is the time in samples to propagate from one end of the string to the other and back, or the total "string loop" delay. The loop delay is also equal to twice the number of spatial samples along the string. A digital simulation diagram for the terminated ideal string is shown in Fig. 10. A "pick-up"is shown at the arbitrary location $x = \xi$.

The Ideal Plucked String

The ideal plucked string (Morse 1936) is defined as an initial string displacement and a zero initial velocity distribution. In general, the initial displacement along the string $y(0,x)$ and the initial velocity distribution $y(0,x)$ fully determine the resulting motion in the absence of further excitation.

An example of the appearance of the traveling-wave components and the resulting string shape shortly after plucking a doubly terminated string at a point one fourth along its length is shown in Fig. 11. The negative traveling-wave portions can be thought of as inverted reflections of the incident waves, or as

Fig. 13. Initial conditions for the ideal plucked string when the wave variables are chosen to be proportional to acceleration or curvature. If the bandlimited ideal pluck position is centered on a spatial sample, there is only a single nonzero sample in each of the initial delay lines.

Fig. 14. Initial conditions for the ideal struck string in a velocity wave simulation. Since $v^{\pm} = \pm f^{\pm}/R = Fcy^{1\pm}$, the initial velocity distribution can be integrated with respect to X from X = 0, divided by C, and negated in the upper rail to obtain equivalent initial displacement waves (Morse 1936).

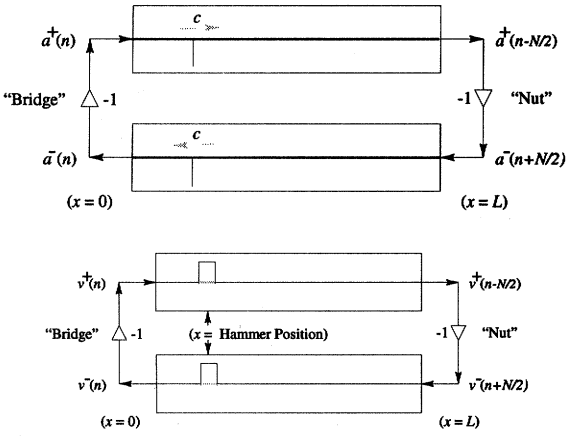
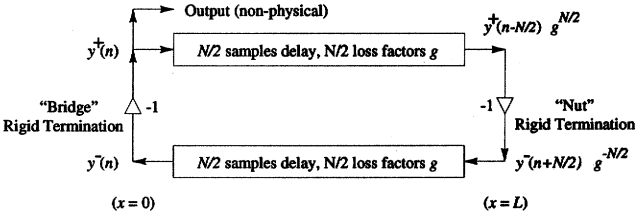


Fig. 15. Discrete simulation of the rigidly terminated string with resistive losses. The N loss factors g are embedded between the delay-line elements.



doubly flipped "images" that are coming from the other side of the terminations.

An example of an initial "pluck" excitation in a digital waveguide string model is shown in Fig. 12. One fine point to note for the discrete-time case is that we cannot admit a sharp corner in the string since that would have infinite bandwidth, which would alias when sampled. Therefore, for the discrete-time case, we define the ideal pluck to consist of an arbitrary shape, as in Fig. 12, lowpass filtered to half the sampling rate (or less). Alternatively, we can simply require the initial displacement shape to be bandlimited to spatial frequencies less than $f_s/2c$. Since all real strings have some degree of stiffness that prevents the formation of perfectly sharp corners, and since real plectra are never in contact with the string at only one point, and since the frequencies we do allow span the full range of human hearing, the bandlimited restriction is not a binding one. So, given proper bandlimiting of the initial shape, it is valid to replace the continuous curve with its samples without changing the information content.

Note that acceleration (or curvature) waves are a simple choice for plucked string simulation, since the ideal pluck corresponds to an initial impulse in the delay lines at the pluck point. Of course, since we require a bandlimited excitation, the initial acceleration distribution will be replaced by the impulse response of the antialiasing filter. If the antialiasing filter chosen is the ideal lowpass filter cutting off at $f_s/2$, the initial acceleration $a(0,x) \doteq \ddot{y}(0,x)$ for the ideal pluck becomes

$$a(0,x) = \frac{A}{X}\,\mathrm{sinc}\left(\frac{x - x_p}{X}\right),$$

where A is amplitude, x_p is the pick position, and $\mathrm{sinc}[(x - x_p)X]$ is the ideal, bandlimited impulse, centered at x_p and having a rectangular spatial frequency response extending from $-\pi X$ to πX. Division by X normalizes the area under the initial shape curve. If x_p is chosen to lie exactly on a spatial sample $x_m = mX$, the initial conditions for the ideal plucked string are as shown in Fig. 13 for the case of acceleration or curvature waves. All initial samples are zero except one in each delay line.

More generally, bowed string models involving a string with a periodic plucking excitation (Smith 1983) can be calibrated to recorded data by means of linear predictive coding (LPC), which has been very successful in speech modeling (Makhoul 1975; Markel and Gray 1976).

The Ideal Struck String

The ideal struck string (Morse 1936) involves a zero initial string displacement but a nonzero initial velocity distribution. In concept, a "hammer strike" transfers an "impulse" of momentum to the string at time 0 along the striking face of the hammer. An example of "struck" initial conditions is shown in Fig. 14 for a striking "hammer" having a rectangular shape.

Fig. 16. Discrete simulation of the rigidly terminated string with consolidated losses (frequency-independent). All N *loss factors* g *have been "pushed" through delay elements and combined at a single point.*

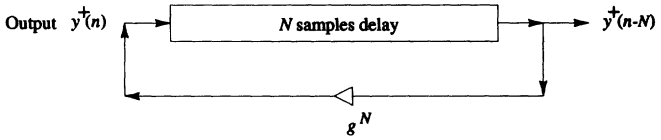
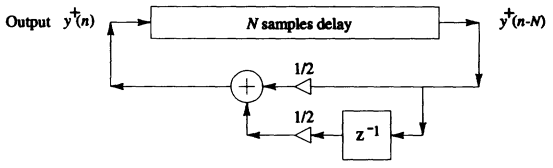


Fig. 17. Rigidly terminated string with the simplest frequency-dependent loss filter. All N *loss factors (possibly including losses due to yielding terminations) have been consolidated at a single point and replaced by a one-zero filter approximation.*



The Damped Plucked String

Without damping, the ideal plucked string sounds more like a cheap electronic organ because the sound is perfectly periodic and never decays. The Fourier transform of the initial "string loop" contents gives the Fourier series for the periodic tone produced, and static spectra are very boring to the ear. Incorporating damping means we use exponentially decaying traveling waves instead of nondecaying waves. As discussed above, it saves computation to lump the loss factors that implement damping in the waveguide in order to minimize computational cost and round-off error.

To illustrate how significant the computational savings can be, consider the simulation of a "damped guitar string "model in Fig. 15. For simplicity, the length L string is rigidly terminated on both ends. Let the string be "plucked" by initial conditions so that we need not couple an input mechanism to the string. Also, let the output be simply the signal passing through a particular delay element rather than the more realistic summation of opposite elements in the bidirectional delay line.

In this string simulator, there is a loop of delay containing $N = 2L/X = f_s/f_1$ samples where f_1 is the desired pitch of the string. Because there is no input/output coupling, we may lump all of the losses at a single point in the delay loop. Furthermore, the two reflecting terminations (gain factors of –1) may be commuted so as to cancel them. Finally, the rightgoing delay may be combined with the leftgoing delay to give a single, length N, delay line. The result of these inaudible simplifications is shown in Fig. 16.

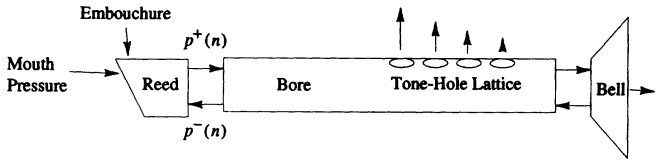
If the sampling rate is $f_s = 50$ kHz and the desired pitch is $f_1 = 100$ Hz , the loop delay equals $N = 500$ samples. Since delay lines are efficiently implemented as circular buffers, the cost of implementation is normally dominated by the loss factors, each one requiring a multiply every sample, in general. (Losses of the form $1 - 2^{-k}$, $1 - 2^{-k} - 2^{-1}$, etc., can be efficiently implemented using shifts and adds.) Thus, the consolidation of loss factors has reduced computational complexity by three orders of magnitude, that is, by a factor of 500 in this case. However, the physical accuracy of the simulation has not been compromised. In fact, the accuracy is improved because the N round-off errors arising from repeated multiplication by g have been replaced by a single round-off error in g^N.

The Plucked String with Frequency-Dependent Damping

As discussed above, damping should increase at higher frequencies for better realism. This means the loss factors g of the previous section are replaced by digital filters having gains that decrease with frequency and never exceed 1. These filters commute with delay elements because they are time invariant. Thus, following the reasoning of the previous section, they can be lumped at a single point in the digital waveguide. Let $\hat{G}(z)$ denote the resulting string loop filter. We have the constraint $|\hat{G}(e^{j\omega T})| \leq 1$, and making it zero or linear phase will restrict consideration to symmetric FIR filters only.

In the simplest case of a first-order lowpass loss filter, $\hat{G}(z) = b_0 + b_1 z^{-1}$, the linear-phase requirement imposes $b_0 = b_1$. Assuming the damping approaches zero at frequency zero implies that $b_0 = b_1 = 1$. Thus, two equations in two unknowns uniquely determine the coefficients to be $b_0 + b_1 = 1/2$, which gives a string loop frequency response equal to $\hat{G}(e^{j\omega T}) = \cos(\omega T/2)$, $|\omega| \leq \pi f_s$.

Fig. 18. A schematic model for single-reed woodwinds.



The simulation diagram for the ideal string with the simplest frequency-dependent loss filter is shown in Fig. 17. Readers of the computer music literature will recognize this as the structure of the Karplus-Strong algorithm (Karplus and Strong 1983; Jaffe and Smith 1983; Sullivan 1990). The Karplus-Strong algorithm, per se, is obtained when the initial conditions used to "pluck" the string are obtained by filling the delay line with random numbers, or "white noise." We know the initial shape of the string is obtained by adding the upper and lower delay lines of Fig. 15; that is, $y(t_n,x_m) = y^+(n-m) + y^-(n+m)$. It was also noted above that the initial velocity distribution along the string is determined by the difference between the upper and lower delay lines. Thus, in the Karplus-Strong algorithm, the string is "plucked" by a completely random initial displacement and initial velocity distribution. This is a very energetic excitation, and usually in practice a lowpass filter is applied to the white noise to subdue it.

Single-Reed Instruments

A simplified model for a single-reed instrument is shown in Fig. 18. If the bore is cylindrical, as in the clarinet, it can be modeled quite simply using a bidirectional delay line. If the bore is conical, as in the saxophone, it can still be modeled as a bidirectional delay line, but interfacing to it is slightly more complex (Benade 1988; Smith 1991). Because the main control variable for the instrument is air pressure in the mouth at the reed, it is convenient to choose pressure waves for the waveguide variables.

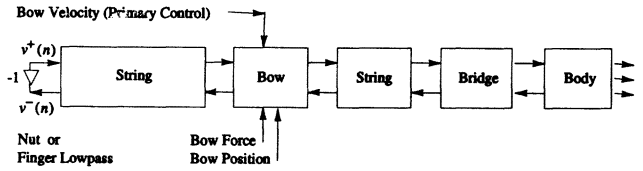
To a first order, the bell passes high frequencies and reflects low frequencies, where "high" and "low" are relative to the diameter of the bell. Thus, the bell can be regarded as a simple "crossover" network, as is used to split signal energy between woofers and tweeters in loudspeakers. Tone holes can be treated similarly. However, much better tone-hole models exist in the literature (Keefe 1982), and they can be adapted to the traveling-wave formulation in a straightforward way.

Since the length of the clarinet bore is only a quarter wavelength of the fundamental (in the lowest, or "chalumeau" register), and since the bell diameter is much smaller than the bore length, most of the sound energy reflects back into the bore. The low-frequency energy that makes it out of the bore radiates in a fairly omnidirectional pattern. Very high-frequency traveling waves do not "see" the enclosing bell and pass right through it, radiating in a more directional beam. The directionality of the beam is proportional to how many wavelengths fit along the bell diameter; in fact, many wavelengths away from the bell, the radiation pattern is proportional to the two-dimensional spatial Fourier transform of the exit aperture, a disk at the end of the bell (Morse and Ingard 1968).

The theory of the single reed has been described by McIntyre, Schumacher, and Woodhouse (1983), and an efficient waveguide synthesis technique based on that theory has already been described (Smith 1986). Essentially, if the reed mass is neglected, its effect can be implemented using a single table-lookup or segmented polynomial evaluation per sample whose input is the difference between mouth pressure and incoming bore pressure, and whose output is the reflection coefficient "seen" at the mouthpiece inside the bore. The lookup table (or polynomial) appears as a nonlinear termination of the bore, presenting a signal-dependent (and mouth-pressure dependent) reflection coefficient. The player's embouchure controls damping of the reed, reed aperture width, and other parameters, and these can be implemented as parameters on the contents of the lookup table or nonlinear function.

Instruments of the brass family have also been convincingly simulated using this general approach (Cook 1991). In the brass model, the "lip reed" is modeled as a second-order mass-spring system whose mass corresponds to the mass of the lips and whose spring constant corresponds to lip tension. Damping is a third control. A waveguide flute has also been implemented (Karjalainen et al. 1991).

Fig.19. A schematic model for bowed-string instruments.



Bowed Strings

A general block diagram for bowed strings is shown in Fig. 19. The bow divides the string into two sections, so the bow model is a nonlinear two-port, in contrast with the reed, which is a one-port terminating the bore at the mouthpiece. In the case of bowed strings, the primary control variable is bow velocity, so velocity waves are the natural choice for the delay lines.

The theory of bow-string interaction has been described (McIntyre and Woodhouse 1979; McIntyre, Schumacher, and Woodhouse 1983). The basic operation of the bow is to reconcile the bow-string friction curve with the string state and string wave impedance. In a bowed-string simulation as in Fig. 19, a velocity input (which is injected equally in the left-going and rightgoing directions) must be found such that the transverse force of the bow against the string is balanced by the reaction force of the moving string. If bow-hair dynamics are neglected, the bow-string interaction can also be simulated using a memoryless table lookup or segmented polynomial (Smith 1986). An overall model for the violin is developed elsewhere (Smith 1983).

Conclusion

Starting with the traveling-wave solution to the wave equation and sampling across time and space, we obtained a modeling framework known as the "digital waveguide" approach. Its main feature is computational economy in the context of a true physical model. Successful computational models have been obtained for musical instruments of the string, wind, and brass families, and more are on the way.

Physics-based synthesis can provide extremely high quality and expressivity in a very compact algorithm; however, new models must be developed for each new kind of instrument, and for many instruments, no sufficiently concise algorithm is known. Sampling synthesis, on the other hand, is completely general since it involves only playing back and processing natural recorded sound. However, sampling synthesis demands huge quantities of memory for the highest quality and multidimensional control. It therefore seems reasonable to expect that many musical instrument categories now being implemented via sampling synthesis will ultimately be upgraded to sleek, computational models based on musical acoustics. As this evolution proceeds, the traditional instrument quality available from a given area of silicon should increase dramatically.

References

Benade, A. H. 1988. "Equivalent Circuits for Conical Waveguides." *Journal of the Acoustical Society of America* 83:1764–1769.

Cook, P. R. 1991. "TBone: An Interactive WaveGuide Brass Instrument Synthesis Workbench for the NeXT Machine." In *Proceedings of the International Computer Music Conference*. San Francisco: International Computer Music Association, pp. 297–300.

Cremer, L. 1984. *The Physics of the Violin*. Cambridge Massachusetts: MIT Press.

Jaffe, D., and J. O. Smith. 1983. "Extensions of the Karplus-Strong Plucked String Algorithm." *Computer Music Journal* 7:56–69.

Karjalainen, M., U. K. Laine, T. Laakso, and V. Valimaki. "Transmission-Line Modeling and Real-Time Synthesis of String and Wind Instruments." In *Proceedings of the International Computer Music Conference*. San Francisco: International Computer Music Association, pp. 293–294.

Karplus, K., and A. Strong. 1983. "Digital Synthesis of Plucked String and Drum Timbres." *Computer Music Journal* 2(7):43–55.

Keefe, D. H. 1982. "Theory of the Single Woodwind Tone Hole" and "Experiments on the Single Woodwind Tone Hole." *Journal of the Acoustical Society of America* 72:676–699.

Loy, N. J. 1988. *An Engineer's Guide to FIR Digital Filters*. Englewood Cliffs, New Jersey: Prentice Hall.

Makhoul, J. 1975. "Linear Prediction: A Tutorial Review." In *Proceedings of the IEEE* 63:561–580.
Markel, J. D., and A. H. Gray. 1976. "Linear Prediction of Speech." New York: Springer-Verlag.
McIntyre, M. E., and J. Woodhouse. 1979. "On the Fundamentals of Bowed String Dynamics." *Acustica* 43:93–108.
McIntyre, M. E., R. T. Schumacher, and J. Woodhouse. 1983. "On the Oscillations of Musical Instruments." *Journal of the Acoustical Society of America* 74:1325–1345.
Morse, P. M. 1936. *Vibration and Sound*. Published by the American Institute of Physics for the Acoustical Society of America, Woodbury, New York USA,1976 (1st ed. 1936; 2nd ed. 1948).
Morse, P. M., and K. U. Ingard. 1968. *Theoretical Acoustics*. New York: McGraw-Hill.
Parks, T. W., and C. S. Burrus. 1987. *Digital Filter Design*. New York: Wiley.
Rabiner, L. R., and B. Gold. 1975. *Theory and Application of Digital Signal Processing*. Englewood Cliffs, New Jersey: Prentice Hall.
Smith, J. O. 1983. "Techniques for Digital Filter Design and System Identification with Application to the Violin." Ph.D. diss., Electrical Engineering Department, Stanford University.
Smith, J. O. 1985a. "Introduction to Digital Filter Theory." In J. Strawn, ed., *Digital Audio Signal Processing: An Anthology*. Los Altos, California: William Kaufmann. A shortened version appears in C. Roads, ed., 1989. *The Music Machine*. Cambridge, Massachusetts: MIT Press.
Smith, J.O. 1986. "Efficient Stimulation of the Reed-Bore and Bow-String Mechanisms." In *Proceedings of the International Computer Music Conference*. San Francisco: International Computer Music Association.
Smith, J. O. 1987. "Music Applications of Digital Waveguides" (a compendium containing four related papers and presentations). CCRMA Technical Report STAN-M-67. Stanford University. (All CCRMA Technical Reports can be obtained by calling 415-723-4971. There is also a master bibliography available.)
Smith, J. O. 1990. "Efficient Yet Accurate Models for Strings and Air Columns Using Sparse Lumping of Distributed Losses and Dispersion." In *Proceedings of the Colloquium on Physical Modeling*.
Smith, J. O. 1991. "Waveguide Simulation of Non-Cylindrical Acoustic Tubes." In *Proceedings of the International Computer Music Conference*. San Francisco: International Computer Music Association, pp. 304–307.
Smith, J. O., and P. Gossett. 1984. "A Flexible Sampling-Rate Conversion Method." In *Proceedings of the IEEE Conference on Acoustics, Speech and Signal Processing*. New York: IEEE Press 2:19.4.1–19.4.2.
Sullivan, C. R. 1990. "Extending the Karplus-Strong Algorithm to Synthesize Electric Guitar Timbres with Distortion and Feedback." *Computer Music Journal* 14:26-37.

Appendix: Programming Example for the Plucked String

```
/*
 * pluck.c
 *
 * Elementary plucked string
 * simulation based on the digital
 * waveguide approach.
 *
 * J.O. Smith, June 6, 1992
 */
#include <libc.h>

/*
 * Set the sampling rate (Hz).
 */
#define SRATE 44100

/*
 * Delay-line data structure.
 * The delay-line descriptor
 * consists of a base address "data",
 * the delay-line length in shorts,
 * a pointer to the current place
 * in the delay line (initially 0),
 * and a pointer to the last element
 * which is used to wrap the pointer.
 * We use 16-bit shorts to make the
 * simulation more like what would
 * be implemented in VLSI or in a
 * digital signal processing (DSP)
 * chip.
 */
typedef struct _DelayLine {
  short *data;
  int length;
  short *pointer;
  short *end;
} DelayLine;

/*
 * dl = initDelayLine(length);
 *
 * Delay-line initialization routine.
 * Allocates memory and initializes all
 * pointers.
 */
static DelayLine
 *initDelayLine(int len) {
  DelayLine *dl = (DelayLine *)
        malloc(sizeof(DelayLine));
  dl->length = len;
  if (len > 0)
    dl->data = (short *)
          malloc(len * sizeof(short));
  else
    dl->data = 0;
  dl->pointer = dl->data;
  dl->end = dl->data+len-1;
  return dl;
 }

/*
 * freeDelayLine(dl);
 *
 * Deallocates memory reserved for
 * delay lines
 */
static void
freeDelayLine(DelayLine *dl) {
 if (dl && dl->data)
   free(dl->data);
 dl->data = 0;
 free(dl);
}

/* If your compiler does not recognize
    the "inline" keyword, you should
    uncomment the following line: */
/* #define inline */

/*
 * setDelayLine(dl,values,scale);
 *
 * Initialize delay-line contents
 * a double array normalized to
 * lie between -1.0 and 1.0. A
 * scale factor may also be
 * specified.
 */
inline static void
setDelayLine(DelayLine *dl,
     double *values, double scale) {
 int i;
 for (i=0; i < dl->length; i++)
    dl->data[i] = (short)
       (32768.0 * scale * values[i]);
}

/*
 * outsamp = dl_update(dl, insamp);
 *
 * This is the normal operation of the
 * delay line. You pass it the sample
 * going into the delay line, and it
 * returns the sample coming out.
 * Internally, the circular buffer
 * pointer is incremented and wrapped
 * if necessary.
 */
static inline short
dl_update(DelayLine *dl, short insamp) {
 short *ptr = dl->pointer;
 short outsamp = *ptr;
 *ptr++ = insamp;
 if (ptr > dl->end)
    ptr = dl->data;
 dl->pointer = ptr;
```

```
 return outsamp;
}

/*
 * sample = dl_access(dl, position);
 *
 * Returns the delay-line sample at
 * an offset "position" samples into
 * delay-line's past.
 */
static inline short
dl_access(DelayLine *dl,
    int position) {
 short *outloc
       = dl->pointer - position;
 while (outloc < dl->data)
    outloc += dl->length;
 while (outloc > dl->end)
    outloc -= dl->length;
 return *outloc;
}

/*
 * Allocate two global delay-line
 * pointers to comprise a digital
 * waveguide for the ideal string.
 */
static DelayLine
       *upper_rail,
       *lower_rail;

/*
 * flipArray(doublesArray, length);
 *
 * Reverses the elements of an array
 * of type double.
 */
static inline void
 flipArray(double *a, int n) {
 int i,j,nh=n/2;
 double temp;
 for (i=0; i<nh; i++) {
    j = n-i-1;
    temp = a[i];
    a[i] = a[j];
    a[j] = temp;
 }
}

/*
 * initString(amp,pitch,pick,pickup);
 *
 * Initialize the string corresponding
 * to a pluck at the string point
 * "pick," normalized between 0.0
 * and 1.0, where 0 is at the bridge and
 * 1 is at the nut. "amp" is the
 * initial displacement under the pick.
 * The string is adjusted to vibrate at
 * frequency "pitch" Hz (ignoring delays
 * in the filter at the bridge). The
 * string output position is set to
 * "pickup" (also normalized 0 to 1).
 */
static inline int
initString(double amplitude,double pitch,
    double pick, double pickup){
 int i;
 int rail_length = SRATE/pitch/2;
 int pickPoint
    = MAX(rail_length * pick,1);
 double upslope = amplitude/pickPoint;
 double downslope = amplitude
       / (rail_length-pickPoint-1);
 double initial_shape[rail_length];

 upper_rail = initDelayLine(rail_length);
 lower_rail = initDelayLine(rail_length);

 for (i=0; i<pickPoint; i++)
    initial_shape[i] = upslope * i;
 for (i=pickPoint; i<rail_length; i++)
    initial_shape[i]
       = downslope * (rail_length-1-i);

 /*
 * Initial conditions for the
 * ideal plucked string.
 * "Past history" is measured
 * from the end of the array.
 */
 setDelayLine(lower_rail,
          initial_shape, 0.5);
 flipArray(initial_shape, rail_length);
 setDelayLine(upper_rail,
          initial_shape, 0.5);

 return pickup * rail_length;
}

/*
 * freeString();
 *
 * Deallocate the two delay-lines
 * comprising the string.
 */
static inline void freeString(void) {
 freeDelayLine(upper_rail);
 freeDelayLine(lower_rail);
}

/*
 * outsamp = bridgeReflection(insamp);
 *
 * Implement a reflection at the bridge.
 * "insamp" is the sample of the
 * traveling wave impinging on the
 * bridge, and "outsamp" is returned
```

```
 * as the reflected traveling wave
 * sample. In principle, this filter
 * should have a frequency response
 * given by [Rb(f)-R]/[Rb(f)+R], where
 * Rb(f) is the driving-point impedance
 * of the bridge at frequency f, and
 * R is the characteristic impedance
 * of the string (geometric mean of
 * tension and density). However, in
 * this simple example, the reflection
 * filter is taken to be a simple
 * one-pole filter with coefficients
 * chosen to eliminate multiplies.
 */
static inline short
bridgeReflection(int insamp) {
          /* filter memory: */
 static short state = 0;
 /*
  * Implement a one-pole lowpass
  * filter with feedback
  * coefficient = 0.5
  * (no multiplies).
  */
 short outsamp = (state >> 1) + (insamp >> 1);
 state = outsamp;
 return outsamp;
}

/*
 * outsamp = nextSample(pickup);
 *
 * Returns the next output sample
 * from the string simulator at
 * position "pickup" along the
 * string (from 0 to 1).
 * initString() must be called
 * first.
 */
static inline short
nextSample(int pickup_loc) {
 short yp0,ym0,ypM,ymM;
 short outsamp;

 /* Output at pickup location */
 outsamp = dl_access(upper_rail,
                pickup_loc);
 outsamp += dl_access(lower_rail,
                -pickup_loc);

 /* Wave traveling into "bridge": */
 ym0 = dl_access(lower_rail,0);

 /* Wave to the "nut": */
 ypM = dl_access(upper_rail,
             upper_rail->length-1);

 /* Inverting reflection
    at rigid nut: */
 ymM = -ypM;

 /* Filtered reflection
    at yielding bridge: */
 yp0 = -bridgeReflection(ym0);

 /* String state update */
 dl_update(upper_rail, yp0);
 dl_update(lower_rail, ymM);

 return outsamp;
}

/*
 * writeSound(name,data,length);
 *
 * Utility for writing a mono sound
 * to a soundfile on the NeXT Computer
 * "name" is the soundfile name created,
 * "data" contains the 16-bit array
 * of shorts to be written, and
 * "length" is the number of shorts.
 */
#import <sound/sound.h>
static int
writeSound(char *name, short *soundData,
    int sampleCount) {
 int i, err;
 short *data;
 SNDSoundStruct *sound;
 SNDAlloc(&sound,
       sampleCount * sizeof(short),
       SND_FORMAT_LINEAR_16,
       SRATE,1,4);
 data = (short *) ((char *)sound
             + sound->dataLocation);
 for (i = 0; i < sampleCount; i++)
    data[i] = soundData[i];
 err = SNDWriteSoundfile(name,sound);
 if(err)
    fprintf(stderr,
       " Could not write sound file %s\n",name);
 else
    printf("File %s written.\n",name);
 return err;
}

/*
 * writeString();
 *
 * Writes the state of the string
 * to a soundfile. The right-going
 * traveling-wave along the string is
 * written to "upper.snd", the left-
 * going wave is written to "lower.snd",
 * and the current string displacement
 * (the sum of the traveling waves) is
 * written to "string.snd".
 */
static void writeString(void){
 int i;
 int sampleCount = upper_rail->length;
```

```
  short data[sampleCount];

  for (i = 0; i < sampleCount; i++)
     data[i] = dl_access(upper_rail,i);
  writeSound("upper.snd",data,
              sampleCount);

  for (i = 0; i < sampleCount; i++)
     data[i] = dl_access(lower_rail,-i);
  writeSound("lower.snd",data,
              sampleCount);

  for (i = 0; i < sampleCount; i++)
     data[i] = dl_access(upper_rail, i)
             + dl_access(lower_rail, -i);
  writeSound("string.snd",data,
              sampleCount);
}

/*
 * Main program.
 *
 * Command-line arguments are parsed,
 * the string is initialized, and the
 * simulation is run for the requested
 * duration. The output soundfile
 * can be played to hear the quality
 * of the simulation, and if the
 * "writeSamp" argument is nonnegative,
 * three string snapshot files are
 * written (as sound files) which can
 * be displayed to view the string
 * and traveling wave shapes at that
 * time instant.
 */
void main (int argc, char *argv[]) {
 int i, sampleCount;
 short *data;
 double amp,duration,pitch;
 double pick,pickup,writesample;
 int pickupSample;

 if (argc != 8) {
    fprintf(stderr, "Usage: "
       "%s amp(<1.0) "
       "pitch(Hz) "
       "pickPosition(<1.0) "
       "pickupPosition(<1.0) "
       "duration(sec) "
       "writeSamp "
       "out.snd\n",
       argv[0]);
    fprintf(stderr, "example: "
       "%s .5 100 .1 .2 1 -1 "
       "test.snd\n", argv[0]);
    exit(1);
    }

    sscanf(argv[1],"%lf",&amp);
    sscanf(argv[2],"%lf",&pitch);
    sscanf(argv[3],"%lf",&pick);
    sscanf(argv[4],"%lf",&pickup);
    sscanf(argv[5],"%lf",&duration);
    sscanf(argv[6],"%lf",&writesample);

    sampleCount = duration * SRATE;
    data = (short *)
          malloc(sampleCount
                * sizeof(short));

    pickupSample = initString(amp,
                pitch, pick, pickup);
    for (i=0; i<sampleCount; i++) {
       if (i==writesample) {
          printf("Writing string "
                "snapshot at "
                "sample %d\n",i);
          writeString();
       }
       data[i]
          = nextSample(pickupSample);
    }
    writeSound("test.snd", data,
                sampleCount);
    freeString();
    exit(0);
}
```