Introduction to Computer Music

Week 11 Physical Models

Version 1, 2 Nov 2018

Roger B. Dannenberg

Topics Discussed: **Mass-Spring Model, Karplus-Strong Plucked String Algorithm, Waveguide Model, Commuted Synthesis, Electric Guitar Model, Analysis Example**

1 Introduction

One promising way to create sounds is to simulate or model an acoustic instrument. If the model is accurate, then the details of control and vibration can in principle lead to realistic sounds with all the control possibilities of physical instruments. This approach might be called physics-based modeling, but the common terminology is simply *physical models*. Physical models can be contrasted with *abstract synthesis* or the use of mathematical functions (such as FM synthesis and Additive synthesis), *sampling*, and *source/filter models*. None of these alternative approaches really capture the complexities of physical systems. When aspects of physical systems defy analysis, we can resort to simulation to compute and predict the behavior of those systems. However, even simulation is selective and incomplete. The key is to model the interesting aspects while keeping the overall simulation and its computation tractable.

Like all of the synthesis methods we have covered, physical modeling is not one specific technique, but rather a variety of related techniques. Behind them all, however, is the basic idea that by understanding how sound / vibration / air / string behaves in some physical system (an instrument), we can model that system in a computer and thereby generate realistic sounds through computation.

2 Mass-Spring Model

A simple example of a physical model is the Mass-Spring model consisting of a string and a set of masses attached to the string (shown in Figure 1). This is a discrete approximation of a continuous string where mass is distributed uniformly throughout the length of the string. By “lumping” the mass at a finite set of points, we can use digital simulation to model the string. In the model, we consider the string between masses to be mass-less springs that pull on the masses with a force that is proportional to the stretch of the spring.

To analyze all the forces here: the springs are pulling on the objects in opposite directions, the masses at the ends are assumed fixed. Because the springs are pulling in both directions, there is little/no longitudinal force on the objects, but there is a restoring force. So when the string is bent up with the concave side facing down, some of the forces on the masses are downward, as shown by "Restoring Force" in Figure 1. Conversely, when the string is down with the concave side facing up, the net force is pulling up. These forces will accelerate the objects. If we put the string in this configuration and release it, then the left half will accelerate downward and right half would go upward. As the masses are pulled to zero displacement, or to a straight line between the end points, there is no more net force on the objects but the masses will keep moving and stretch the string in the opposite direction until the restoring force can slow them down and reverse the direction. This motion will repeat, causing the string to oscillate.

This is a computationally expensive model because you have to compute the force on each one of the masses and store the velocity and position of the masses for each time step of the simulation. But computers are fast, and discrete time simulation is mostly multiplies and adds, so you can easily run interesting models (including this one)

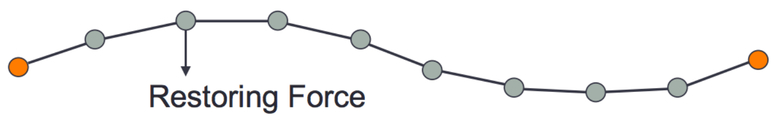


Figure 1: Mass-Spring Model of a String

in real-time. The number of modes (partials) that you can support corresponds to the number of masses. Also, you can add stiffness and other interesting properties into the string, e.g. the string can be non-linear, it can have a driving force, there can be friction, etc.

3 Karplus-Strong Plucked String Algorithm

Let's take a look at a variation of the Mass-Spring model. This is a really simple but very effective physical model of a plucked string, called the *Karplus-Strong algorithm* (so named for its principal inventors, Kevin Karplus and Alex Strong). One of the first musically useful physical models (dating from the early 1980s[1]), the Karplus-Strong algorithm has proven quite effective at generating a variety of plucked-string sounds (acoustic and electric guitars, banjos, and kotos) and even drumlike timbres. Nyquist has an implementation in the function `pluck`.

Here's a simplified view of what happens when we pluck a string: At first the string is highly energized and it vibrates, creating a fairly complex (meaning rich in harmonics) sound wave whose fundamental frequency is determined by the mass and tension of the string. Gradually, thanks to friction between the air and the string, as well as the dissipation of energy in the form of sound waves, the string's energy is depleted. The higher frequencies tend to lose energy the fastest, so the wave becomes less complex as it decays, resulting in a "purer" tone with fewer harmonics. After some amount of time all of the energy from the pluck is gone, and the string stops vibrating.

If you have access to a stringed instrument, particularly one with some very low notes, give one of the strings a good pluck and see if you can see and hear what's happening based on the description above.

3.1 How a Computer Models a Plucked String with the Karplus-Strong Algorithm

Now that we have a physical idea of what happens in a plucked string, how can we model it with a computer? The Karplus-Strong algorithm does it like this: first we start with a buffer full of random values—noise. (A buffer is just some computer memory (RAM) where we can store a bunch of numbers.) The numbers in this buffer represent the initial energy that is transferred to the string by the pluck. The Karplus-Strong algorithm looks like this:

$$Y_t = \frac{1}{2}(Y_{t-p} + Y_{t-p-1})$$

Here, p is the period or length of the buffer, t is the current sample count, and Y is the output of the system.

To generate a waveform, we start reading through the buffer and using the values in it as sample values. If we were to just keep reading through the buffer over and over again, we would get a complex, periodic, pitched waveform. It would be complex because we started out with noise, but pitched because we would be repeating the same set of random numbers. (Remember that any time we repeat a set of values, we end up with a pitched sound.) The pitch we get is directly related to the size of the buffer (the number of numbers it contains) we're using, since each time through the buffer represents one complete cycle (or period) of the signal.

[1] When I was an undergraduate at Rice University, a graduate student was working with a PDP-11 mini-computer with a vector graphics display. There were digital-to-analog converters to drive the display, and the student had connected them to a stereo system to make a primitive digital audio system. I remember his description of the synthesis system he used, and it was exactly the Karplus-Strong algorithm, including initializing the buffer with random numbers. This was in the late 1970s, so it seems Karplus and Strong *reinvented* the algorithm, but certainly deserve credit for publishing the work.

Now, here's the trick to the Karplus-Strong algorithm: each time we read a value from the buffer, we average it with the last value we read. It is this averaged value that we use as our output sample. (See Figure 2.) We then take that averaged sample and feed it back into the buffer. That way, over time, the buffer gets more and more averaged (this is a simple filter, like the averaging filter described in an earlier chapter). Let's look at the effect of these two actions separately.

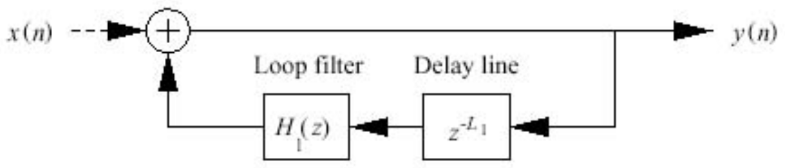


Figure 2: Schematic view of a computer software implementation of the basic Karplus-Strong algorithm. For each note, the switch is flipped and the computer memory buffer is filled with random values (noise). To generate a sample, values are read from the buffer and averaged. The newly calculated sample is both sent to the output stream and fed back into the buffer. When the end of the buffer is reached, we simply wrap around and continue reading at the beginning. This sort of setup is often called a circular buffer. After many iterations of this process, the buffer's contents will have been transformed from noise into a simple waveform. If you think of the random noise as a lot of energy and the averaging of the buffer as a way of lessening that energy, this digital explanation is not all that dissimilar from what happens in the real, physical case. Thanks to Matti Karjalainen for this graphic.

3.2 Averaging and Feedback

First, what happens when we average two values? Averaging acts as a low-pass filter on the signal. Since high frequencies have a high rate of change, averaging has a bigger effect on high frequencies than low ones. So, averaging a signal effectively reduces high frequencies.

The "over time" part is where feeding the averaged samples back into the buffer comes in. If we were to just keep averaging the values from the buffer but never actually putting the average back into the buffer, then we would be stuck with a static waveform. We would keep averaging the same set of random numbers, so we would keep getting the same results.

Instead, each time we generate a new sample, we store it back into the buffer. That way our waveform evolves as we move through it. The effect of this low-pass filtering accumulates over time, so that as the string "rings," more and more of the high frequencies are filtered out of it. The filtered waveform is then fed back into the buffer, where it is filtered again the next time through, and so on. Figure 3 illustrates how the contents of the Karplus-Strong buffer changes and decays over time. After enough times through the process, the signal has been averaged so many times that it reaches equilibrium—the waveform is a flat line and the string has died out.

Physical models generally offer clear, "real world" controls that can be used to play an instrument in different ways, and the Karplus-Strong algorithm is no exception: we can relate the buffer size to pitch, the initial random numbers in the buffer to the energy given to the string by plucking it, and the low-pass buffer feedback technique to the effect of air friction on the vibrating string.

4 Waveguide Model

Now, we consider another model of the string, called the *waveguide model*, introduced by Julius Smith. In a real string, waves travel down the string until they reach the end where the wave is reflected and travels back in the opposite direction. A vibrating string is actually a wave travelling up and down the string, reflecting at both ends, and the left-going and right-going waves sum through superposition to determine the displacement of the string at any given location and time.

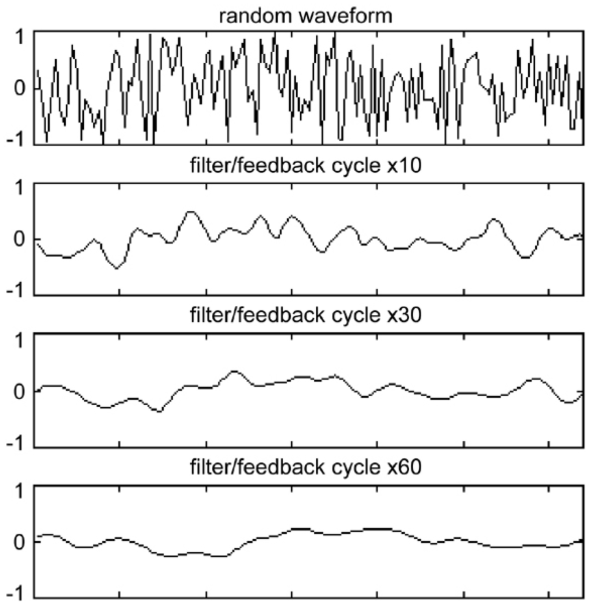


Figure 3: Applying the Karplus-Strong algorithm to a random waveform. After 60 passes through the filter/feedback cycle, all that's left of the wild random noise is a gently curving wave. The result is much like what we described in a plucked string: an initially complex, periodic waveform that gradually becomes less complex over time and ultimately fades away.

It is easy to model wave travel in one direction: we simply delay the samples by storing incoming samples in an array (wrapping around when we reach the end), and reading out older samples. This allows a delay up to the length of the array. If we ignore friction and other losses, a string carrying a waveform can be modeled as a delay.

To model left-going *and* right-going waves, we simply use two one-way models, i.e. two delays. The output of one delay connects to the input of the other. (See Figure 4.) If we want the amplitude of the string at some particular point, we access *both* delays (the left-going and right-going wave models) and sum the amplitudes.

4.1 "Lumped" Filters

Now, what about losses? In a continuous string, there should be loss at every step of the way through the string. Since this would be computationally expensive, we use a shortcut and compute the total losses for the entire trip from one end of the string to the other. This can be expressed as a filter that we can apply to the output of the delay. This so-called "lumped" filter is efficient and can give the same effect as accumulating tiny losses at each sample of delay.

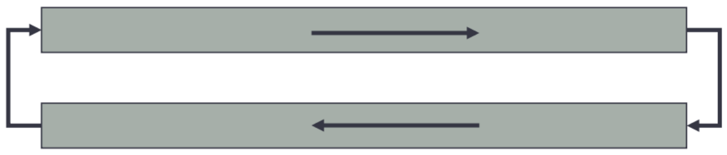


Figure 4: A waveguide model. The right-going and left-going wave, which are superimposed on a single string, are modeled separately as simple delays. The signal connections at the ends of the delays represent reflections. A waveguide can also model a column of air as in a flute.

5 Mechanical Oscillator

To make a sustained sound, we must overcome the losses in the waveguide. The case of the bowed string is probably the easiest to understand, so we will start there. Figure 5 illustrates the oscillation in a bowed string. The bow alternately sticks to and slips across the string. Rather than reaching a steady equilibrium where the bow pulls the string to some steady stretched configuration, the "slip" phase reduces friction on the the string and allows the string to move almost as if were plucked. Interestingly, string players put rosin on the bow, which is normally sticky, but when the string begins to slide, the rosin heats up and a molecular level of rosin liquifies and lubricates the bow/string contact area until the string stops sliding. It's amazing to think that rosin on the string and bow can melt and re-solidify at audio rates!

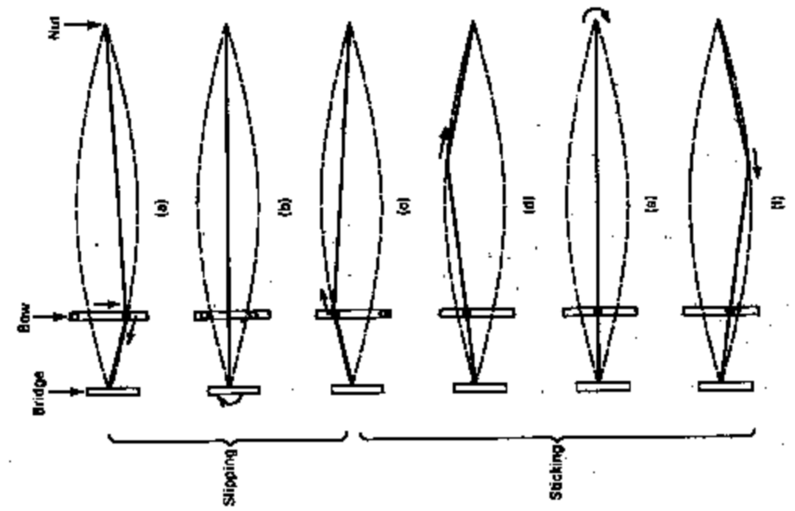


Figure 5: A bowed string is pulled by the bow when the bow sticks to the string. At some point the bow does not have enough friction to pull the string further, and the string begins to slip. The sliding reduces the friction on the string, which allows it to stretch in opposition to the bowing direction. Finally, the stretching slows the string and the bow sticks again, repeating the cycle. (From http://physerver.hamilton.edu/courses/Fall12/Phy175/ClassNotes/Violin.html)

5.1 McIntyre, Woodhouse (1979) + Schumacher (1983)

An important advance in physical models came from McIntyre and Woodhouse who resorted to models to help understand the nature of oscillation in acoustical instruments. Later, Schumacher, a physics professor at Carnegie Mellon University, visited McIntyre and Woodhouse to learn more about their work and the three physicists wrote a paper that formed the basis for a lot of work in the field of Computer Music.

Their model follows the basic ideas we have outlined so far. Rather than a bi-directional waveguide, they combined the two delays into one as shown in Figure 6, with a single low-pass filter to model losses over the entire loop. Since the model is for a woodwind rather than a bowed string, the delay is considered to represent a traveling pressure wave rather than string displacement. McIntyre, Woodhouse and Schumacher added a non-linear element to generate oscillation.

5.2 Smith: Efficient Reed-Bore and Bow-String Mechanisms (ICMC 86)

Julius Smith was influenced by McIntyre, Woodhouse and Schumacher and developed computer music instruments that model the clarinet and violin.

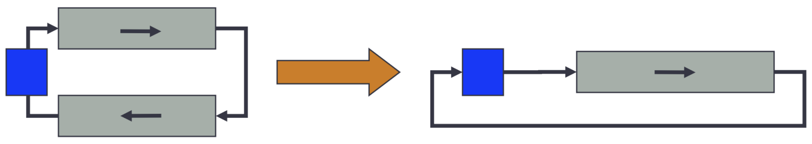


Figure 6: McIntyre Woodhouse model consisting of a delay representing sound traveling through the bore of a clarinet and a filter representing the losses over the round trip. The figure shows that the single delay is equivalent to a bi-directional waveguide with perfect reflection at one end.

Figure 7 shows Smith's clarinet model. It includes a waveguide with low-pass filter (-LP in the figure). At the left side of the figure is a model of the reed, which has non-linear behavior that enables oscillation. A clarinet *reed* is a thin, flat, flexible plate that vibrates over an opening to the clarinet's body or *bore*. The reed acts as a valve to let air in when the reed is up or open, and to block the air when the reed is down or closed.

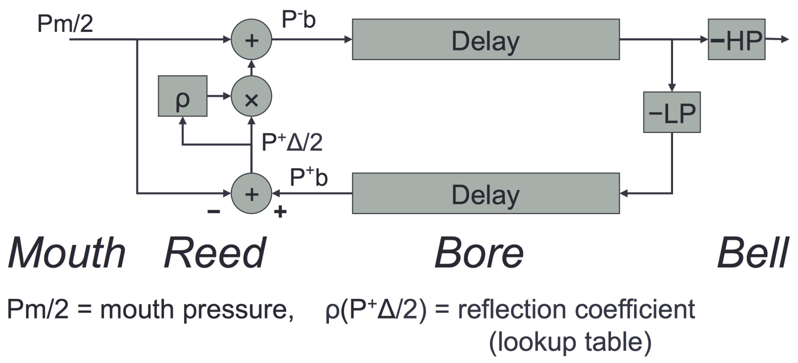


Figure 7: Clarinet model

To understand oscillation in the clarinet model, we can follow the "story" of one period. To being with, the reed is open and pressure from the mouth enters the clarinet. The pressure also closes the reed valve, at least to some extent. The high pressure front travels to the bell, the flare at the end of the clarinet, where the pressure is reflected. The reflection is inverted, so now the pressure front is negative. The negated pressure wave returns to the reed, where it is reflected again. This time, the reflection does not invert the wave because this end is effectively closed. The negative pressure acts to close the reed even further. The negative pressure returns to the bell, is inverted again and reflects back to the reed as a positive pressure wave. This positive pressure tends to open the reed, allowing air through the reed valve, which reinforces the positive pressure wave, and another cycle begins. If the addition of energy or pressure compensates for losses, a sustained oscillation will result.

Figure 8 illustrates a bowed string model. In this model, the bow is not at the end of the string, so there is one waveguide from the bow to the bridge (where strings are anchored over the body) and one waveguide from the bow to the nut (where strings are anchored at the end of the fingerboard). The bow has a non-linear element (ρ in the figure) that models the change in friction between the stick and slip phases.

Also, the model includes a filter between the bridge and the output. In a violin, the bridge transfers vibration from strings to the body, and the body radiates sound into the room. The body has resonances and radiates in a frequency-dependent manner, so a filter to model the transfer of sound from bridge to room is important to getting a violin-like sound.

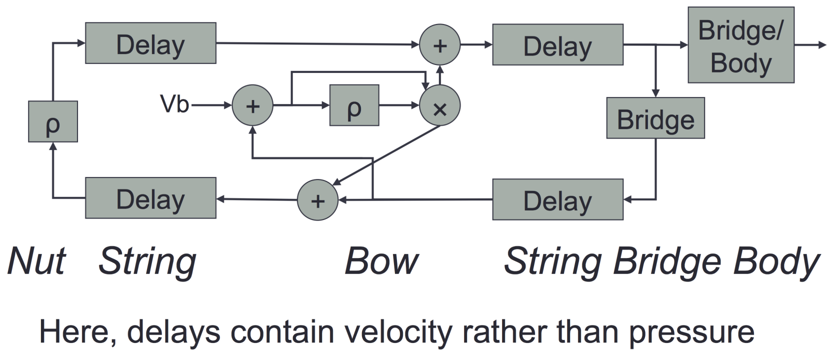


Figure 8: Bowed String Model

6 Flute Physical Model

Figure 9 is a simple model for a flute, showing a single delay that models the round-trip through the flute, a low-pass filter (LP) to model the losses, and a high-pass filter (HP) to model radiation into the room.

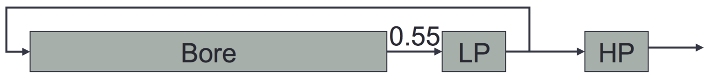


Figure 9: Flute Physical Model

Figure 10 shows a more elaborate model that includes a mouthpiece to drive sustaining oscillation. The input to the mouthpiece is the sum of a smooth pressure envelope (the breath) and some random noise (turbulence). As with other models, there must be some non-linearity or the model will simply settle into a steady state. In this case, the non-linearity is $x - x^3$, which is simple but enough to allow oscillation to occur.

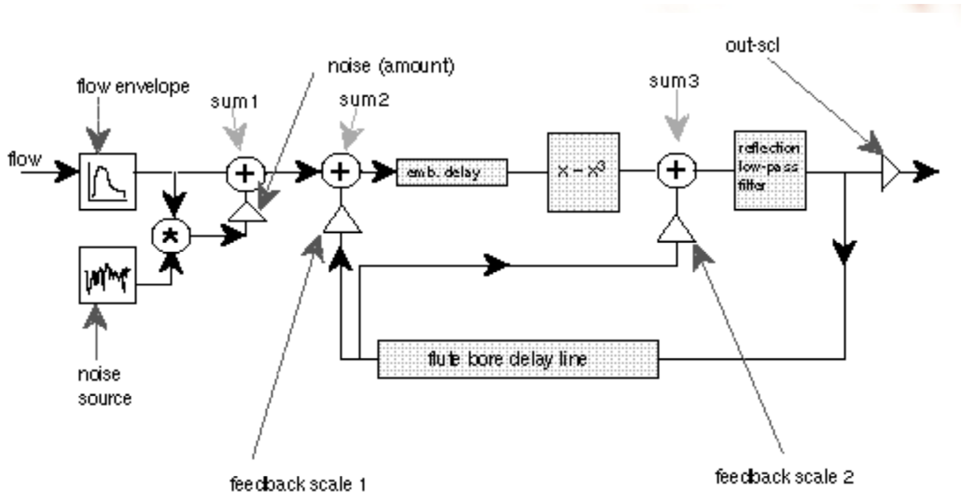


Figure 10: Flute Physical Model

7 Physical Models in Nyquist

Nyquist has a number of built-in physical models. Many of them come from the Synthesis Tool Kit (STK).

`pluck(`*`pitch, dur, final-amp`*`)` is an extended Karplus-Strong plucked string model. The extension inserts a filter into the loop (besides the simple averaging filter we learned about) to allow sub-sample delays needed for accurate tuning. In addition, the rate of decay can be modified by the optional parameters *`dur`* and *`final-amp`*. Increasing either or both parameters lowers the decay rate. You might want to multiply by an envelope to avoid a click at the end if the *`final-amp`* is high.

`clarinet(`*`step, breath-env`*`)` is a basic STK clarinet model. There are several variations on this model in Nyquist that allow continuous control over frequency, breath envelope, vibrato, reed-stiffness, and noise through additonal parameters. See the Nyquist Reference Manual for details.

`sax(`*`step, breath-env`*`)` is a basic STK saxophone model (called "saxophony"). As with `clarinet`, there are variations with additional parameters.

8 Commuted Synthesis

One of the problems with physical models of guitars, violins, and pianos is that vibrating strings excite a complex 3-dimensional body that is computationally hard to simulate. We can assume that the body is a kind of complex filter. We can characterize the body by tapping it at the bridge or point where the string is attached and measuring the impulse response.

Now, one way to model the body is to simply convolve the string force with the body's impulse response. In other words we just filter the string with the body filter and we are done. But convolution is expensive, so researchers thought of another approach.

Consider a piano model with a hammer model that transmits force to a string model, the string model transmits force to a bridge, and the body model filters the bridge force to obtain an output. Now, *the string and body are both just filters*! Multiplication and convolution and linear filters are all commutative (it's all more-or-less the same thing), so we can *switch the order of the string and body filters*. This makes no sense physically, but in our simulation, instead of driving the string with an impulse (as if hit by a hammer), we can drive the string with the impulse response of the body! Thus, for every note, we just have to "play" the impulse response into the string model, saving the need for a complex body filter on the output.

For strings, the commuted synthesis model is a little more complex because the bow is repeatedly pumping energy into the string. We need to run a bowed string model to detect when the bow slip occurs. Then, we treat that as the driving impulse, and every time the bow slips, we drive a *second* string model with the violin body impulse response. We take the output from this second string model.

9 Electric Guitar Model

Charles R. Sullivan developed an interesting guitar model[2]. His work was motivated more by obtaining usable control than by being a faithful model.

The basic model is shown in Figure 11, and you can see that this is closely related to the Karplus-Strong model. The low-pass filter determines the decay rate of the string and can also change the effective length (and frequency) of the string by inserting additional delay. In this model, an FIR filter is used:

$$y_n = a_0 x_n + a_1 x_{n-1} + a_2 x_{n-2}$$

but this potentially has gain at zero Hz (DC).

[2]Charles R. Sullivan, "Extending the KarplusStrong Algorithm to Synthesize Electric Guitar Timbres with Distortion and Feedback." Computer Music Journal, Vol. 14, No. 3, Fall 1990.

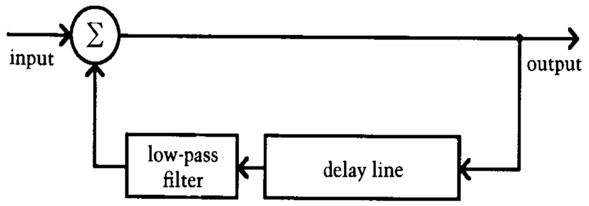


Figure 11: Electric Guitar Model by Charles R. Sullivan. The model is based on Karplus-Strong, but the low-pass filter is customized and the model allows continuous input through the summation node to allow for plucking while the string is still vibrating and feedback.

9.1 Loop Filter Design

To eliminate DC, we can add a high-pass filter:

$$y_n = a_0 x_n + a_1 x_{n-1} + b_1 y_{n-1}$$

We also want to provide continuous tuning, for which we need a sub-sample delay. Simple linear interpolation:

$$y_n = c_0 x_n + c_1 x_{n-1}$$

can be used, but this also produces attenuation (low-pass filter), so we can adjust the loop filter (FIR) to provide only the additional attenuation required.

After all this, the model is still not perfect and might require a compensating boost at higher frequencies, but Sullivan decided to ignore this problem: Sometimes higher frequencies will suffer, but the model is workable.

9.2 Tuning and Glissandi

For tuning, we can just round the desired delay length to an integer number of samples and use interpolation to add the remaining fractional length. To achieve *glissando*, where the pitch changes continuously, we slowly change c_0, c_1 in the interpolator. When one coefficient reaches 1, we can change the delay length by 1, flip c_0, c_1, and there is no glitch, but we are ready to continue the glissando.

However, changing the loop length will require a change in the loop FIR filter. It is expensive to recalculate all the filters every sample, so Sullivan updates the filters once per period. There may be small artifacts, but these will generate harmonics that are masked by the string harmonics.

9.3 Distortion

In electric guitars, distortion of a single note just adds harmonics, but distortion of a sum of notes is not the sum of distorted notes: distortion is not linear, so all those nice properties of linearity do not apply here.

Sullivan creates distortion using a soft clipping function so that as amplitude increases, there is a gradual introduction of non-linear distortion. The signal is x and the distorted signal is $F(x)$ in the following equation, which is plotted in Figure 12:

$$F(x) = \begin{cases} \frac{2}{3} & x \geq 1 \\ x - \frac{x^3}{3} & -1 < x < 1 \\ -\frac{2}{3} & x \leq -1 \end{cases}$$

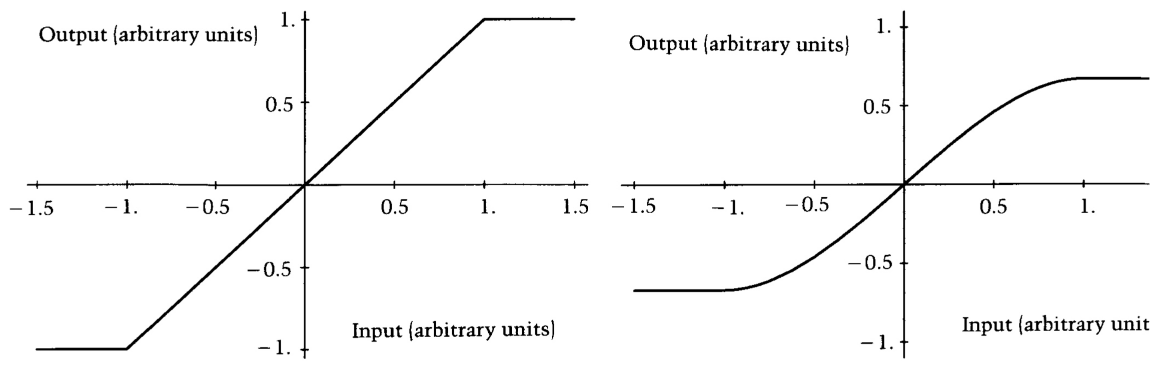


Figure 12: Distortion functions. At left is "hard clipping" where the signal is unaffected until it reaches limits of 1 and -1, at which points the signal is limited to those values. At right is a "soft clipping" distortion that is more like analog amplifiers with limited output range. The amplification is very linear for small amplitudes, but diminishes as the signal approaches the limits of 1 and -1.

9.4 Feedback

A wonderful technique available to electric guitarists is feedback, where the amplified signal is coupled back to the strings which then resonate in a sustained manner. Figure 13 illustrates Sullivan's configuration for feedback. There are many parameters to control gain and delay. Sullivan notes that one of the interesting things about synthesizing guitar sounds with feedback is that even though it is hard to predict exactly what will happen, once you find parameter settings that work, the sounds are very reproducible. One guiding principle is that the instrument will tend to feedback at periods that are multiples of the feedback delay. This is quite different from real feedback with real guitars, where the player must interact with the amplifier and guitar, and particular sounds are hard to reproduce.

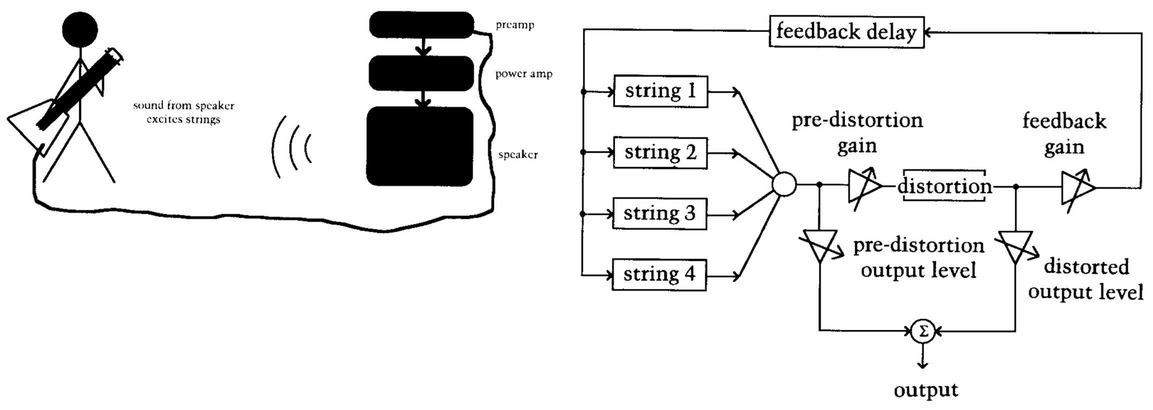


Figure 13: Feedback is achieved by feeding some of the output through a delay and back into each string model.

9.5 Initializing the String

When plucking a guitar string in this model, how should we initialize the string? In the Karplus-Strong model, strings are just initialized with random numbers, but this is not necessary, and it can be more interesting to simulate plucking as displacing the string at a particular point and releasing it. Plucking closer to the end of the string gives a brighter sound (try it if you have a guitar)! Figure 14 (a) shows the desired geometry of the initial string configuration. In the light of what we know about waveguides, this initial position must be split as right- and left-going waves as shown in Figure 14 (b). If there is a single delay (as in Karplus-Strong), we need to concatenate the right- and left-going wave configurations, resulting in an initial value as shown in Figure 14 (c).

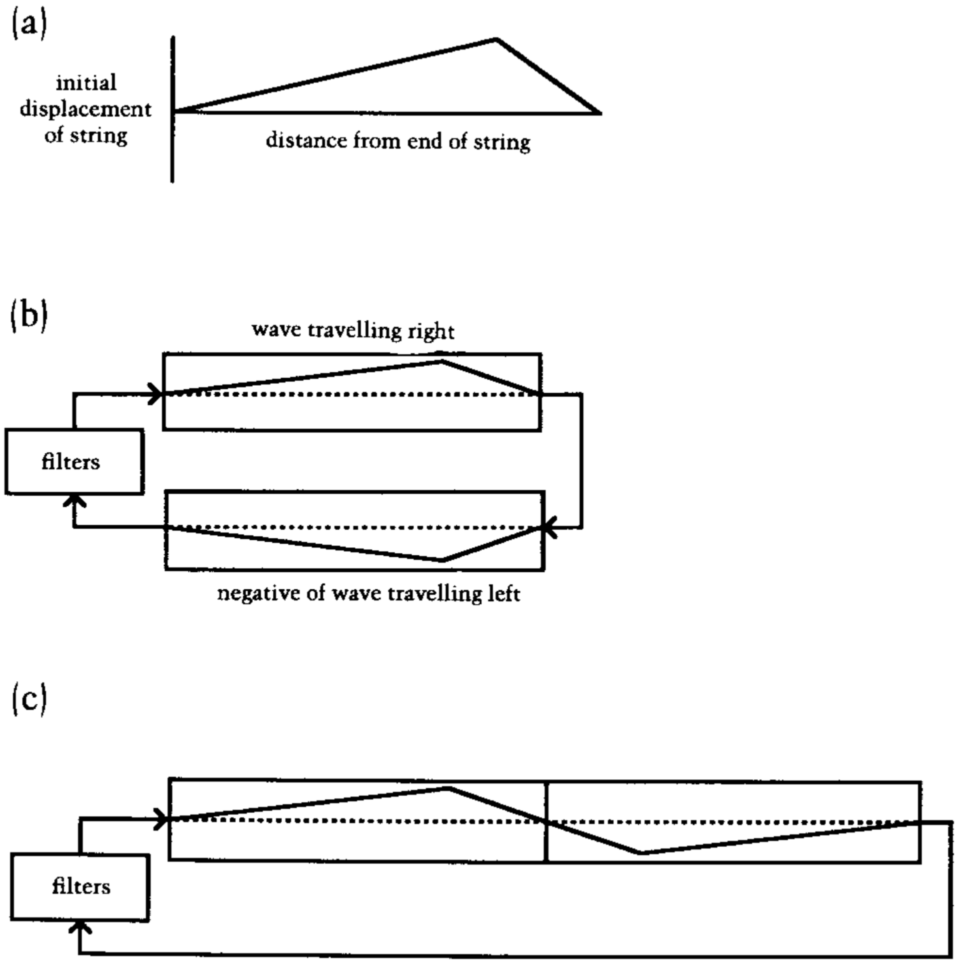

(a)
initial displacement of string
distance from end of string
(b)
wave travelling right
filters
negative of wave travelling left
(c)
filters

Figure 14: Initializing the string. The physical string shape (a) contains both left- and right-going waves (b), so we need to combine them to get a full round-trip initial waveform (c).

9.6 Additional Features

Sullivan's electric guitar model is interesting because it shows how models can be extended incrementally to incorporate various features, either for study or for additional sound generation and control. There are still more things one could do with this model, including:

- Adding guitar body resonances,
- Coloration and distortion of guitar amplifiers,
- Effects processors, including:
  - Distortion,
  - Wah-wah pedals,
  - Chorus, etc.

10 Analysis Example

One of the challenges of physical models is that we need many parameters to make the model behave like real acoustic instruments. In some cases, parameters are obtained by trial-and-error, or else calculated, e.g. given a desired pitch, we can calculate the length of a waveguide.

However, it is interesting to estimate parameters from real sounds, and doing this enables us to check on whether the model is really capturing the behavior of the acoustic instrument. We present an example of the analysis of acoustic guitar sounds to estimate physical model parameters. This example should give some idea of how parameter estimation can be approached. The word *estimation* should be emphasized—we rarely get the chance to measure anything exactly or directly.

In Figure 15, we see a plot of the amplitudes of harmonics of a plucked guitar string as they decay over time. Since decay is mainly due to losses as the wave travels up and down the string, we can fit lines to the curves and estimate the loss as a function of frequency. (Note that the decay should be exponential, so by plotting on a dB (log) scale, the decays appear as straight lines.)

After measuring the decay at different frequencies, we can fit a filter to the data, as shown in the lower half of the figure. If the filter is simple as shown, the fit will not be perfect, but measurement errors will tend to average out and the overall trend of the filter is likely to be well-estimated. Alternatively, one could fit a complex filter exactly to all the data points, but this would run the risk of "overfitting," or incorportaing measurement errors into the model.

10.1 Driving Force

Another part of the model we might like to measure is the driving force on the string. After fitting a filter to the string recording, we can create an inverse filter, apply that to the recording, and end up with a "residual" that represents the input. Then, we can drive the string model with the residual to get a realistic sound.

11 2D Waveguide Mesh

The 1-dimensional waveguide can be extended to 2 or 3 dimensions. This adds a lot of computation, but allows us to model plates and drums in the 2-D case, and resonant chambers and wind instruments in the 3-D case.

Figure 16 shows a 2-D waveguide mesh and the modeled propagation of a wave over a surface in work by Van Duyne and Smith.

12 Summary

Physical models simulate physical systems to create sound digitally. A common approach is to model strings and bores (in wind instruments) with recirculating delays, and to "lump" the losses in a filter at some point in the loop. Non-linear elements are added to model how a driving force (a bow or breath) interacts with the wave in the recirculating delay to sustain an oscillation. Digital waveguides offer a simple model that separates the left- and right-going waves of the medium.

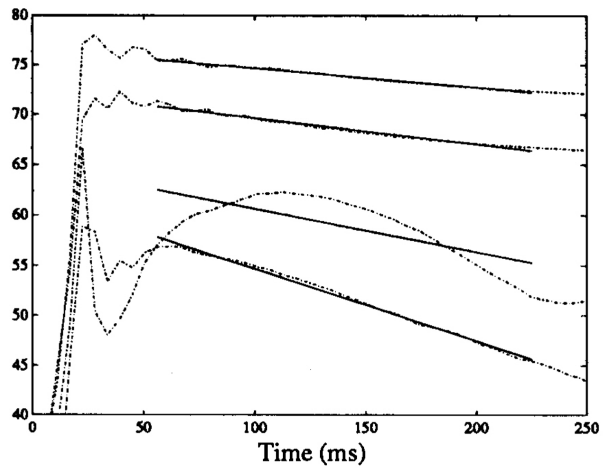


Fig. 7 ***Temporal envelopes of the four lowest harmonics of a guitar tone and straight lines fits. The amplitude scale is in dB.***

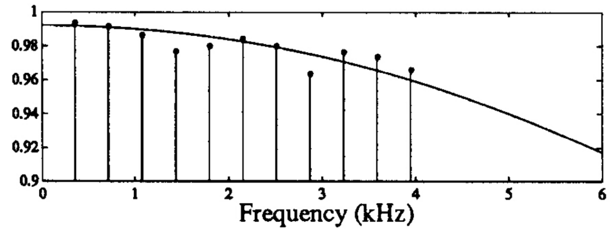


Fig. 8 ***Estimated magnitude spectrum (circles) and magnitude response of a 1st-order IIR filter.***

Figure 15: Analysis Example, from Karjalainen, Valimaki, and Janosy. "Towards High Quality Sound Synthesis of the Guitar and String Instruments" in Proc. ICMC 1993.

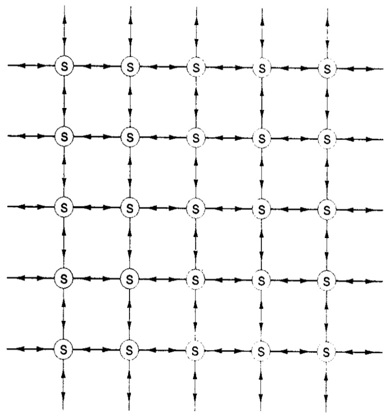
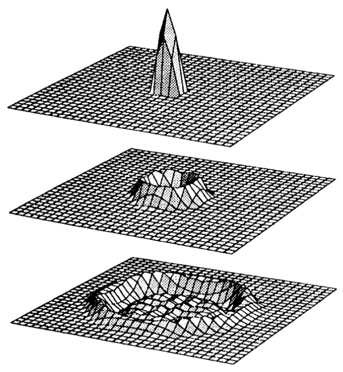


Figure 3. The 2-D Digital Waveguide Mesh



From: Van Duyne and Smith, "Physical Modeling with the 2-D Digital Waveguide Mesh," in Proc. ICMC 1993.

Figure 16: 2D Waveguide Mesh and some some simulation results.

12.1 Advantages of Physical Modeling

One advantage of physical models is that non-linear vibrating systems have complex behaviors. Simulations can create complex and interesting behaviors that tend to arise naturally from models. In spite of potentially complex behavior, physical models tend to have a relatively small set of controls that are meaningful and intuitively connected to real-world phenomena and experience. Models also tend to be modular. It is easy to add coupling between strings, refine a loop filter, etc. to obtain better sound quality or test theories about how instruments work.

12.2 Disadvantages of Physical Models

On the other hand, the real 3-D world resists simplifications. For example, violin bodies are very complex and perceptually important. When simplifications break down, physical model computation becomes very high. For example, there are experiments using 3-D waveguides models of brass instruments that run on supercomputers much slower than real time.

Control is also difficult. Just as real instruments require great skill and practice, we should not expect simple inputs will immediately result in great sounds. It is difficult to invert recorded sounds to determine the control required to produce them. Consider all the muscles and motions involved in playing the violin or trumpet, or the fact that it takes years to become an accomplished performer on these instruments. One answer to this problem is that physical models should form the basis of new instruments that can be controlled in real-time by humans. These instruments might be as difficult to play as acoustic instruments, but they might have interesting new sounds and capabilities.

13 Acknowledgments

Thanks to Shuqi Dai and Sai Samarth for editing assistance.

Portions of this work are taken almost verbatim from *Music and Computers, A Theoretical and Historical Approach* (`http://sites.music.columbia.edu/cmc/MusicAndComputers/`) by Phil Burk, Larry Polansky, Douglas Repetto, Mary Robert, and Dan Rockmore. Other portions are taken almost verbatim from *Introduction to Computer Music: Volume One* (`http://www.indiana.edu/~emusic/etext/toc.shtml`) by Jeffrey Hass. I would like to thank these authors for generously sharing their work and knowledge.