Real Time Synthesis of
Bowed String Timbres

Charlie Q. Robinson
CEFL Music Project
University of Illinois
Urbana, Illinois 61801

Abstract
A real time synthesis of bowed string timbres developed through the use of a computationally efficient model of a bowed string, and a fast general purpose programmable digital signal processor is discussed. Real time performance of computer data is input (eg via MIDI) to a microcomputer which in turn controls the execution of a synthesis algorithm implemented in microcode on the digital sample generating computer.

String Model:
The output from any linear system can be derived as the convolution of its impulse response and the input. For the case of a violin, the impulse response of the string, or Green’s function, could be convolved with a plucking or bowing function to generate violin tones. A much simpler and more efficient method, first proposed by McIntyre and Woodhouse [1], is evident when one considers the physical process of wave propagation on a string. The impulse response of a string consists of a pseudo-periodic train of increasingly rounded pulses which result from the repeated filtering and reflection of the initial impulse traveling in both directions away from the bow. This process can be duplicated by storing each of the travelling wave components in two shift registers with the output of each filtered, reflected, and fed back into the other. A simplified version of this in which just one travelling wave is represented was first implemented by Karplus and Strong [2]. It was found that a single delay line preloaded with random numbers could accurately simulate a plucked string. Additions to this basic scheme, including methods for precise tuning and variable decay, were added by Jaffe and Shih [3]. The Karplus-Strong algorithm was one of the first synthesizer methods simple enough to produce complex sounds with the limited computational capabilities of a microcomputer.
Bow Model:
A method for describing the interaction between bow and string is described in [1]. It can be summarized as follows. With the assumption that the bow applies force to just one point on the string, string velocity is given by

\[ v(t) = 0.5 c T f(v(t)) + v_B(t) \]  \hspace{2cm} (1)

where \( c \) is the wave propagation speed, \( T \) is the string tension, \( f(t) \) is the force applied by the bow, and \( v_B(t) \) is the velocity of the string resulting from previously established traveling waves. \( f(t) \) is a non-linear function of \( v(t) \) as shown in figure 1a. When the point \((v, f)\) lies on the steep negatively sloped portion of the curve, the string is sticking to the bow, otherwise it is slipping.

A graphical solution to (1) first proposed by Friedlander [4] and Keller [5] consists of superimposing the linear force versus velocity curve

\[ f(t) = 2T/c \left| v - v_B \right| \]  \hspace{2cm} (2)

with the friction curve. It can be seen however, that under certain conditions the two curves can intersect in three places (Fig. 1b) resulting in a non-unique solution. Woodhouse and McIntyre resolve this problem by applying the following hysteresis rule. As string velocity increases, the bow remains in slipping mode until it reaches the capture velocity, \( v_C \). The string velocity then jumps to \( v_B \), bow velocity. The string remains in static mode until the string velocity decreases to \( v_C \) at which point the maximum sticking force is exceeded. The string breaks loose and slips at velocity \( v_2 \) (see figure 2). McIntyre and Woodhouse show that the transition states between bowing modes are unstable and that the dashed portions of the curves in figure 2 are traversed rapidly.

\[ \text{Fig. 1 Friction Curve (a), and Friedlander Ambiguity (b).} \]

\[ \text{Fig. 2 Hysteresis Rule for capture (a) and release (b).} \]

126 1987 ICMC Proceedings
We assume this transition time to be instantaneous and approximate the friction curve with linear segments. If this curve is now combined with (2), the string velocity resulting from the bow input, $v(t)$, can be represented in terms of the current string velocity, $v_1$, as shown in figure 3. This is the basic input bowing function used. In the steady state the bowing function should result in the one slip per cycle oscillations characteristic of violins, as noted by Helmholtz [6]. An intuitive understanding of Helmholtz oscillations can be gained by considering the interaction of the bow with the string reflections. The string is initially drawn aside by the bow until the tension on the string exceeds the maximum sticking force, and the string begins to slip back generating a velocity discontinuity of magnitude $v_2$. This velocity discontinuity travels to the bridge and is reflected back toward the bow. The resulting decrease in string speed at the bowing point takes it back into static mode. The new velocity discontinuity of $v_1$ travels to the nut and back and triggers the dynamic mode. The result is a ramp wave with a slow displacement and rapid return with time durations proportional to the distance from the bow to the nut and bridge respectively.

Hardware:

The program is written in microcode for the Platypus, a high speed digital audio signal processor developed by the CERL Music Group at the University of Illinois. The Platypus is a 32 bit computer with the following functional units:

- 1K x 32 data storage registers
- 1M x 16 data memory
- 2k x 80 program memory
- instruction sequencer
- Platypus - 68000 interface
- stereo 16 variable rate D/A and A/D converters with 128 deep FIFO queues
- three multiplier-accumulators

Operation is controlled using the Platypus Assembly Language in which data flow is specified in register transfer notation. Each instruction requires 50 nanoseconds to execute and can specify several actions to occur in parallel.

1987 ICMS Proceedings 1:17
implementation:
A block diagram of the algorithm is shown below (Fig. 4). Delay lines are represented as $z^{-3}$, where $s$ is the delay in samples. Data is stored in the delay lines in terms of string displacement. Thus an input displacement added to the right-going wave at the top of the diagram reaches the nut after a delay of $m$ samples, and input added to the left-going wave at the bottom reaches the bridge after a delay of $b$ samples. At both ends the samples are filtered and fed back into the beginning of the other delay line. Displacement at any point is found by adding together the contributions by the left and right going waves. String length, $m+b$, and bow location can be changed to within one sample point by varying $m$ and $b$. Finer adjustments can be made through the use of all pass filters as discussed in [3].

![Diagram of Violin Model Block Diagram]

Fig. 4 Violin Model Block Diagram

The terminating filters $H_b(z)$ and $H_n(z)$ are second order low pass filters, (i.e. a weighted average of adjacent sample points), requiring two multiplies and one add. For the case of an open string, $H_n(z)$ represents reflection from the nut rather than from the player’s finger, and can be modeled as a lossless reflection requiring just one multiply. The violin body resonances can be simulated by an output filter $H_z(z)$.

Bow input is a non-linear function of bow velocity $v_b$, pressure $P$, and location $b$; string velocity $v_t$, and displacement $u$, at bowing point, and bow mode (i.e. sticking or slipping), as below.

$$v(t) = v_b(t)$$  \hspace{1cm} (3)

$$v(t) = F(v(t)) + v_t(t)$$  \hspace{1cm} (4)

128 1987 ICMC Proceedings
for static and dynamic bow modes, respectively. \( F(\gamma) \) is the amount string velocity is decreased as a result of rubbing of the bow as shown in figure 4. All velocities are represented as displacement per sample time, thus translation from displacement to velocity (or vice versa) requires just one addition.

Conclusion:
Through the use of high speed hardware, software synthesis in real time is now possible. An enhanced Macintosh is used to accept MIDI data (from any MIDI controller) and generate the required parameters for bow speed, pressure and location as well as pitch. These parameters are passed on to the program running on the platypus. The number of voices that can be generated in real time depends on the complexity of the algorithm. For the cases in which body resonances are not considered and only discrete bow locations are allowed, we were able to generate eight voices. If greater complexity is required with many voices, samples can be sent to a hard disk for replay back. While not real time, time savings are still significant when compared to other machines.

References


