NORDIC ACOUSTICAL MEETING
12 - 14 JUNE 1996 HELSINKI

AN IMPROVED DIGITAL WAVEGUIDE MODEL OF A FLUTE WITH FRACTIONAL DELAY FILTERS

Rami Hänninen[1], Vesa Välimäki[2,3]

1 Helsinki University of Technology
Telecommunications and Multimedia Laboratory
Otakaari 1, FIN-02150 ESPOO, Finland
rami.hanninen@hut.fi
2 University of Westminster
School of Electronic and Manufacturing Systems Engineering
115 New Cavendish Street, LONDON W1M 8JS, United Kingdom
3 Helsinki University of Technology
Laboratory of Acoustics and Audio Signal Processing
Otakaari 5A, FIN-02150 ESPOO, Finland
vesa.valimaki@hut.fi

1 INTRODUCTION

Digital waveguide modeling of musical instruments is one of the most popular sound synthesis techniques at present. This methodology has been mainly developed by Smith [1]. Digital waveguide modeling is well suited to synthesis of wind and string instruments, although the same principles have recently been applied to the piano and percussion instruments as well.

In this paper we concentrate on digital waveguide synthesis of the flute, which has been a target of active research during the recent years. The first version of a real-time digital waveguide flute model was introduced by Karjalainen *et al.* [2] and Välimäki *et al.* [3]. Cook has implemented a simplified version of this model [4]. Similar principles for simulating the air reed mechanism of the flute had been used by McIntyre *et al.* [5] and Wawrzynek [6]. More recently also Coltman [7] has reported on a flute simulation model. Newest developments in flute modeling include the works of Chafe [8] and Cook [9]. A more detailed and physically accurate flute model could be constructed based on recent research by Verge [10]. Other research work on flute synthesis using a time-domain model include [11], [12], and [13].

This paper introduces an improved digital waveguide model of a flute which features a full set of finger holes. The model responds correctly to real playing techniques, including the fingering combinations, overblowing and vibrato. Many of the flute model parameters, such as the lengths of delay lines and finger hole filter parameters, have been computed from measurements taken from a real flute.

So far, many implementations have failed to produce realistic flute sound that could be controlled using the same techniques that are used to play a real flute. One of the major problems has been the note transition. Changing the length of the delay lines of the waveguide flute changes the pitch but produces an audible ‘click’ when the internal state of the model abruptly changes. On the other hand, changing the length of the delay lines gradually changes the state smoothly but produces an undesirable pitch-bend effect.

A solution to these problems was introduced by Välimäki *et al.* [14] (see also [15]). The finger holes that are used to control the pitch in a flute must be incorporated into the model. Their position is adjusted using interpolation techniques, which are implemented as fractional delay filters [16]. Our implementation uses fractional delay finger hole filters that match the function, location and size of the finger holes of a real flute. The model demonstrates many properties of a real flute. Modern playing styles, such as singing into the flute or operating finger holes without blowing into the flute, may also be simulated.

2 WAVEGUIDE FLUTE MODEL

The overall structure of the new flute model is nearly the same as that described in [3]. This version however includes multiple simultaneously functioning finger holes. The signal flowgraph of the new, improved flute model is illustrated in Figure 1.

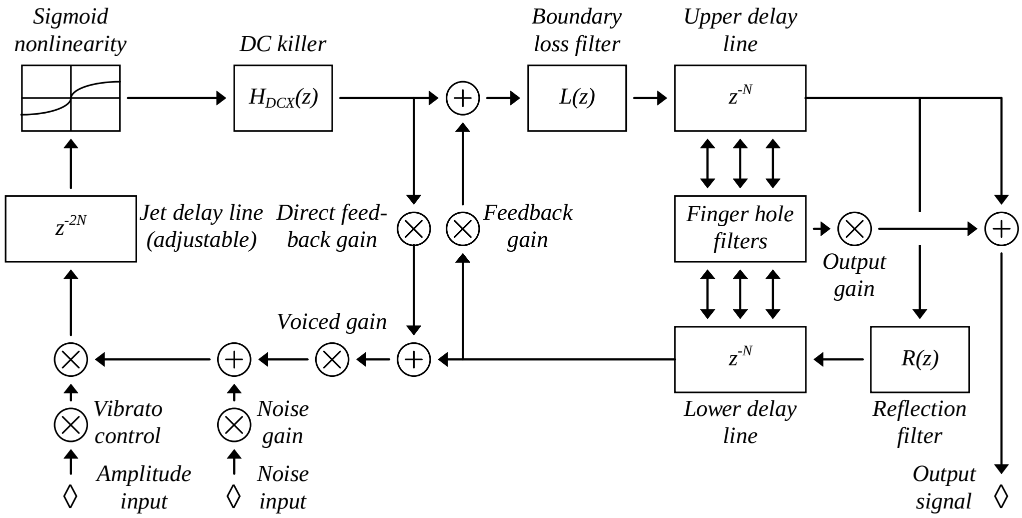

Figure 1. Block diagram of the new waveguide flute model

2.1 Boundary loss filter

The boundary loss filter in [3] has been moved from the end of the upper delay line. When placed before the upper delay line, it now affects also the upper delay line finger hole filter inputs. As explained in [3], the boundary loss filter models signal losses that occur when sound waves propagate along the flute main tube. For the sake of efficiency, all loss filtering is realized by one filter, and as long as there is only one signal output, this is perfectly legal (see [1]). With finger hole filters, a more accurate, but computationally much more expensive solution would be to distribute the loss filtering among several “smaller” loss filters, placed between signal outputs.

2.2 Modeling an open finger hole

Each open finger hole is implemented as a three-port junction. It is assumed that a finger hole is a short side-branch in the bore of the tube. The acoustic impedance of the side-branch is simulated with a first-order IIR filter as described in [14] and [15].

The location of the finger hole must be very carefully adjusted. This is achieved by using interpolation techniques. The input signal to the finger hole model is obtained using some bandlimited interpolation method. (In this work we use Lagrange interpolation). The output signal of the finger hole model must be injected equally carefully back into noninteger points of the delay line. In [17], a new signal processing method, called deinterpolation, was defined which performs this operation. Deinterpolation may be implemented with an FIR filter which has the same coefficients as a corresponding interpolation filter (for further details, see [15]).

3 IMPLEMENTATION

Two different implementations of the flute model have been built: one with Matlab for easy testing and analysis and another with C++ for high performance. This section describes the Matlab implementation since it is simpler and easier to explain. The C++ implementation aims for speed with the cost of simplicity.

3.1 Multiple finger holes

The Matlab version utilizes the matrix operations that the Matlab environment offers to implement the delay lines and fractional delay filters. It turns out that, when properly arranged, the finger hole filtering and the interpolation and deinterpolation operations can be carried out with just a few matrix and vector operations. For example, a delay line of length N can be conveniently expressed as a vector with N components. Using Matlab notation, an empty delay line is constructed like this:

$$DELAYLINE = \mathrm{zeros}(1, N) \tag{1}$$

The first and last components of the vector represent the most recent and the oldest sample in the delay line, respectively. Once initialized, the unit time step operation and insertion of a new sample into the delay line is straightforward:

$$DELAYLINE = \left[NEWSAMPLE \quad DELAYLINE(1:N-1) \right] \tag{2}$$

A traditional — and normally much more efficient — ring-buffer implementation is not practical in an interpreted Matlab environment because the above vectored shifting operation is much simpler and faster than the corresponding ring-buffer version which requires Matlab for-loops and remainder operations[†].

[†] The notion “1:N” defines a sequence 1, 2, ..., N. Given as an argument to a vector, it refers to the vector components 1, 2, ..., N. Note that in Matlab, the first vector component has index one.

Next, we define a sparse *COEFFS* matrix that stores the Lagrange interpolation coefficients for fractional delay filters. *COEFFS* is a $N \times M$ matrix, where N is the length of the delay line and M is the number of finger holes. Each column of the *COEFFS* matrix holds Lagrange coefficients for one fractional delay filter. The coefficient groups are padded with zeros, and the placement of coefficients within columns defines the integer part of each delay. For example, let us assume that $N = 80$, $M = 2$, the order of the fractional delay filters is 3, and the finger holes are located at positions 40.3 and 60.1. The matrix *COEFFS* then becomes:

$$COEFFS = \begin{bmatrix} \underbrace{0 \;\dots\; 0}_{39} & -0.0595 & 0.7735 & 0.3315 & -0.0455 & \underbrace{0 \;\dots\; 0}_{37} \\ \underbrace{0 \;\dots\; 0}_{59} & -0.0285 & 0.9405 & 0.1045 & -0.1045 & \underbrace{0 \;\dots\; 0}_{17} \end{bmatrix}^{\mathrm{T}} \tag{3}$$

where subscript T denotes matrix transposition. After *COEFFS* has been initialized, the input signals for all finger hole models can be computed with a single instruction:

$$VENTINPUT = (UPPER + LOWER) * COEFFS \tag{4}$$

UPPER and *LOWER* are both $1 \times N$ vectors and represent the upper and lower delay lines of the waveguide model, respectively. The *LOWER* delay line flows in the opposite direction than the *UPPER* delay line, and the order of elements in *LOWER* is therefore reversed. *VENTINPUT* is a $1 \times M$ row vector that contains the fractionally delayed samples for the M finger hole models.

The finger hole filters are approximated with first-order IIR filters (see [14] or [15] for details), and the outputs of the M filters are collected into a $1 \times M$ vector:

$$VENTOUTPUT = BR * VENTINPUT - AR * VENTOUTPUT \tag{5}$$

AR and *BR* are finger hole filter parameters and *VENTOUTPUT* on the right side of the equation holds the previous *VENTOUTPUT* values. *AR* and *BR* may be scalars if the parameters are the same for all finger hole filters, or vectors if the parameters vary.

The values in *VENTOUTPUT* are distributed back to the upper and lower delay lines using deinterpolation. The same Lagrange interpolation coefficients that were used to extract the fractionally delayed samples are used to insert them back into the delay lines [15]. Reversing the interpolation process, we first construct a deinterpolation vector:

$$DEINTER = VENTOUTPUT * COEFFS^{\mathrm{T}} \tag{6}$$

This yields a $1 \times N$ vector where each element is the sum over all finger hole filter outputs, weighted by the appropriate fractional delay filter coefficients. These values are then added to the upper and lower delay lines, finishing the deinterpolation operation:

$$\begin{aligned} UPPER &= UPPER + DEINTER \\ LOWER &= LOWER + DEINTER \end{aligned} \tag{7}$$

Listing 1 presents the main loop of the flute model in Matlab. The complete source code may be obtained from [18].

Listing 1. Main loop of the flute model (supporting functions excluded)

```
for I = 1:SAMPLES
  JET(1)       = INPUTAMPL(I) * (PRESSURE(I) + LOOPOUTPUT);  % Jet line
  SIGMOUTPUT   = SIGMOUT*tanh(AMPL(I)*SIGMOFFSET-SIGMIN*JET(JETN));
  TEMP         = SIGMOUTPUT + DCA * DCXOUTPUT1;             % DC killer
  DCXOUTPUT0   = TEMP - DCXOUTPUT1;
  DCXOUTPUT1   = TEMP;
  LOSSINPUT    = DCXOUTPUT0 + BACKGAIN * LOWER(1);    % Boundary losses
  UPPER(1)     = LOSSSCALE*LOSSGAIN*LOSSINPUT + LOSSFREQ*UPPER(2);
  VENTOUTPUT   = BRI * ((UPPER + LOWER) * COEFFS) - ARI * VENTOUTPUT;
  DEINTER      = VENTOUTPUT * COEFFS';                  % Deinterpolation
  UPPER        = UPPER + DEINTER;
  LOWER        = LOWER + DEINTER;
  REFLDELAYY   = RB0*UPPER(N) + RB1*(REFLDELAYX-REFLDELAYY);  % Reflect
  REFLDELAYX   = UPPER(N);
  LOWER(N)     = REFLDELAYY;
  LOOPOUTPUT   = VOICEDGAIN * (LOWER(1) + DIRFDBACK * DCXOUTPUT0);
  JET          = [ 0 JET(1:JETMAXLENGTH-1) ]; % -->  % Move delay lines
  UPPER        = [ 0 UPPER(1:N-1)]; % -->
  LOWER        = [ LOWER(2:N)   0]; % <--
end;
```

4 SIMULATION

The flute model produces a convincing flute-like sound that changes pitch smoothly when the finger holes are opened and closed. Nevertheless, this requires that the finger holes themselves are operated smoothly. This is realistic since the finger holes of a flute do not open or close infinitely fast. Any abrupt change in the values of the finger hole parameters creates a severe discontinuation to the internal state of the model which produces an audible click. Adjusting the parameters gradually removes this effect.

Below we give two examples of the behaviour of the Matlab version of the flute model [18]. Both tests were run using the configurable parameter set shown in Table 1. The only difference between these examples is how the finger hole parameters are operated: in Figure 2, they are changed smoothly, in Figure 3, infinitely fast. The continuous line in the figures shows the state of the finger holes.

Table 1. Test run parameters (FH = Finger Hole)

Flute length:	80 unit delays	Sample rate:	44100 Hz
Number of holes:	2	Simulation time:	1 s
Position of FH1:	40.3	Interpolator degree:	2
Position of FH2:	60.1	Matlab RAND seed:	1

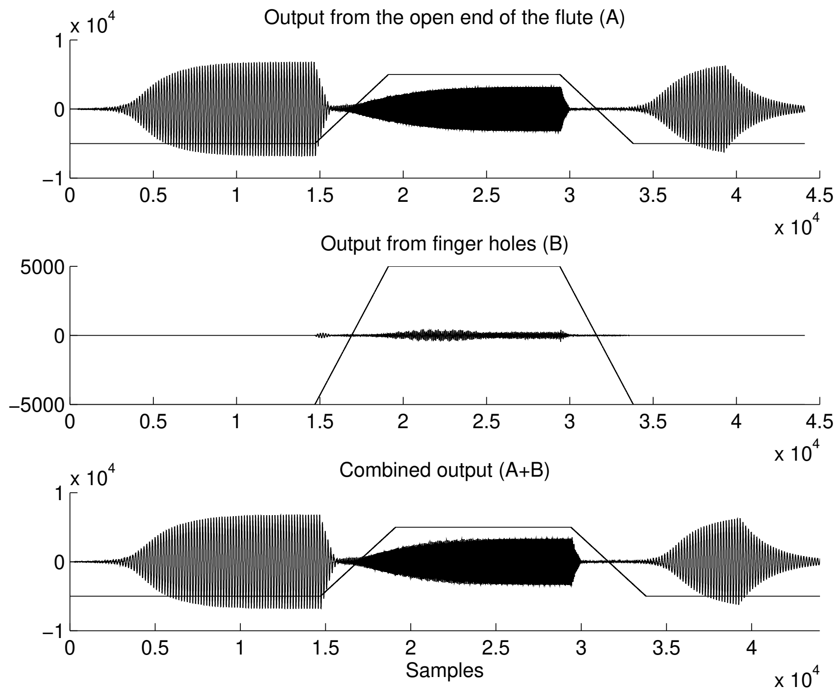

Figure 2. Smooth finger hole operation

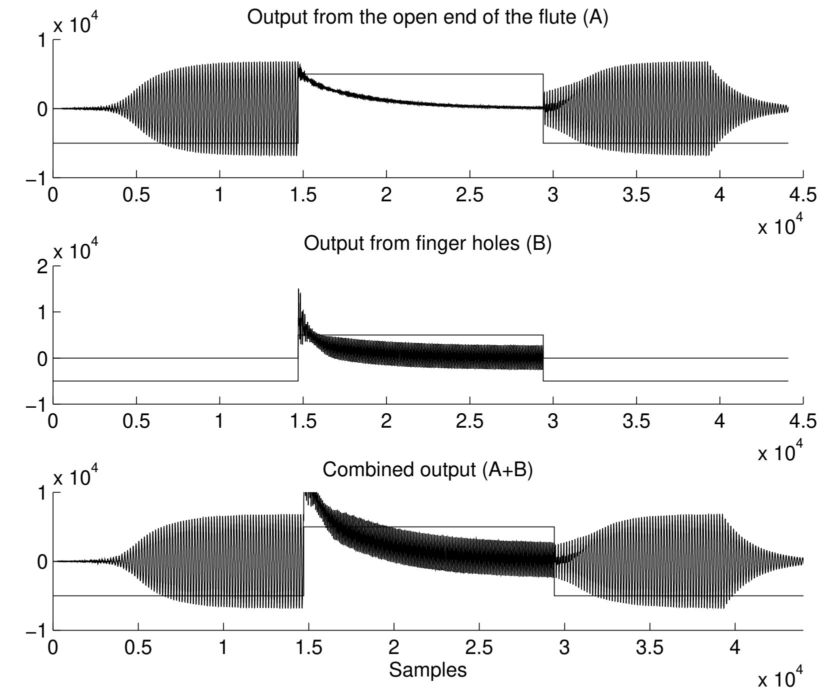

Figure 3. Infinitely fast finger hole operation

A real flute is controlled with finger holes and input air flow. The air flow control is brought into the flute model by controlling the amplitude of the noise input and the length of the jet delay line. The control over the jet delay line length corresponds mainly to the angle in which a flutist blows air into a mouth piece. The blowing technique has the important function of controlling the overblowing effect of the flute. With the flute model, the length of the jet line must match the frequency of the expected sound pitch.

When no overblowing is in effect and all finger holes are closed, the jet line length is equal to the sum of upper and lower line lengths. By halving the length of the jet line, the model starts to overblow, producing one octave higher pitch (the fundamental frequency is doubled). Halving the jet line length again raises the pitch a fifth more, etc.

It is interesting to note that the exact jet line length required to reach a certain note and overblow level depends on the current note and overblowing level of the flute model. The current resonator equilibrium state affects the requirements to reach a new equilibrium state, just like with a real flute. Although this does not usually require any conscious effort, a real flutist must learn the proper blowing techniques for every possible transition from one note to another. With our flute model, the jet line length must be equally carefully adjusted to produce the expected sounds.

5 CONCLUSIONS AND FUTURE PLANS

Our complete implementation of the proposed model includes the full finger hole set of a flute. Measuring the location and sizes of the finger holes from a real flute and converting them to finger hole parameters produces suprisingly good result. Just like when designing and building a flute, many finger hole configurations fail to produce any sound at all. Therefore, finding a set of 15 finger holes that responds correctly to flute fingering combinations is a success in itself. Unfortunately, fine-tuning the numerous parameters of the flute model is a difficult task. An interesting possibility to adjust the parameters of the linear part (tube) of the flute would be to measure the delay and filter characteristics needed for each finger hole, following the guidelines proposed by Karjalainen *et al.* [19]. One more alternative way to adjust the parameter values would be to use an automatic learning system, such as a genetic algorithm [20].

6 REFERENCES

1. Smith, J. O. 1992. Physical modeling using digital waveguides. *Computer Music J.*, vol. 16, no. 4, winter, pp. 74–87.
2. Karjalainen, M., Laine, U. K., Laakso, T. I., and Välimäki, V. 1991. Transmission-line modeling and real-time synthesis of string and wind instruments. *Proc. Int. Computer Music Conf. (ICMC'91)*, Montreal, Canada, Oct. 16–20, 1991, pp. 293–296.
3. Välimäki, V., Karjalainen, M., Jánosy, Z., and Laine, U. K. 1992. A real-time DSP implementation of a flute model. *Proc. 1992 Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP'92)*, San Francisco, California, March 23-26, 1992, vol. 2, pp. 249–252.
4. Cook, P. R. 1992. A meta-wind-instrument physical model, and a meta-controller for real-time performance control. *Proc. Int. Computer Music Conf. (ICMC'92)*, San Jose, California, Oct. 14–18, 1992, pp. 273–276.
5. McIntyre, M. E., Schumacher, R. T., and Woodhouse, J. 1983. On the oscillations of musical instruments. *J. Acoust. Soc. Am.*, vol. 74, no. 5, pp. 1325–1345.

6. Wawrzynek, J. 1989. VLSI models for sound synthesis. In: Mathews, M. V., and Pierce, J. R. (eds.) *Current Directions in Computer Music Research*, MIT Press, Cambridge, Massachusetts, pp. 113–148.

7. Coltman, J. 1992. Time-domain simulation of the flute. *J. Acoust. Soc. Am.*, vol. 92, no. 1, pp. 69–73.

8. Chafe, C. 1995. Adding vortex noise to wind instrument physical models. *Proc. Int. Computer Music Conf. (ICMC'95)*, Banff, Canada, Sept. 3–7, pp. 57–60.

9. Cook, P. R. 1995. Integration of physical modeling for synthesis and animation. *Proc. Int. Computer Music Conf. (ICMC'95)*, Banff, Canada, Sept. 3–7, 1995, pp. 525–528.

10. Verge, M. P. 1995. *Aeroacoustics of Confined Jets, with Applications to the Physical Modeling of Recorder-Like Instruments*. PhD thesis. Eindhoven, The Netherlands, Eindhoven University of Technology. 243 p.

11. Magalotti, R., and Borin, G. 1995. Time-domain simulation of the instruments of the flute family. *Proc. Second Int. Conf. Acoustics and Music Research (CIARM'95)*, Ferrara, Italy, May 19–21, 1995, pp. 61–66.

12. Magalotti, R., Borin, G., and De Poli, G. 1995. A new dipole source for time-domain simulation of flute-like instruments. *Proc. Int. Symp. Musical Acoustics (ISMA'95)*, Le Normont, Dourdan, France, July 2–6, 1995, pp. 491–495.

13. Agulló, J., and Puig, J. 1995. Time-domain modeling of the recorder. *Proc. Int. Symp. Musical Acoustics (ISMA'95)*, Le Normont, Dourdan, France, July 2–6, 1995, pp. 48-53.

14. Välimäki, V., Karjalainen, M., and Laakso, T. I. 1993. Modeling of woodwind bores with finger holes. *Proc. Int. Computer Music Conf. (ICMC'93)*, Tokyo, Japan, Sept. 10–15, 1993, pp. 32–39.

15. Välimäki, V. 1995. *Discrete-Time Modeling of Acoustic Tubes Using Fractional Delay Filters*. Doctoral thesis. Report no. 37. Espoo, Finland, Helsinki University of Technology, Laboratory of Acoustics and Audio Signal Processing. 193 p.

16. Laakso, T. I., Välimäki, V., Karjalainen, M., and Laine, U. K. 1996. Splitting the unit delay—tools for fractional delay filter design. *IEEE Signal Processing Magazine*, vol. 13, no. 1, pp. 30–60.

17. Välimäki, V., Karjalainen, M., and Laakso, T. I. 1993. Fractional delay digital filters. *Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS'93)*, Chicago, Illinois, May 3–6, 1993, vol. 1, pp. 355–358.

18. Hänninen, R, and Välimäki, V, 1995. Matlab source code for the improved waveguide flute model. <URL:ftp://helmholtz.hut.fi/pub/flute/>

19. Karjalainen, M., Välimäki, V., Hernoux, B., and Huopaniemi, J. 1995. Explorations of wind instruments using digital signal processing and physical modeling techniques. *J. New Music Research*, vol. 24, no. 4, pp. 301–317.

20. Vuori, J., and Välimäki, V., 1993. Parameter estimation of non-linear physical models by simulated evolution—application to the flute model. *Proc. Int. Computer Music Conf. (ICMC'93)*, Tokyo, Japan, Sept. 10–15, 1993, pp. 402–404.