Synthesized Flute

Melissa Gibney
Department of Audio and Music Engineering
University of Rochester
Rochester, United States
mgibney@u.rochester.edu

Abstract—**This project tackles the issue of creating a synthesized flute using only physical modeling and discusses the factors contributing to design decisions as well as the technical aspect of the project. While many synthesized flutes exist, very few are physically modeled, as it is far easier to record samples of flute playing than create the instrument from scratch. Physical modeling does have its benefits, such as less memory usage, and it is these benefits that motivated this project to create a realistic flute synthesizer. While the synthesizer that was implemented is not perfect when compared to existing models, it does give insight into how to improve synthesized flutes in the future.**

Keywords—flute, model, synthesizer, air, jet, bore

I. Introduction

The flute is a very versatile instrument capable of performing both fast technical work and slower melodic passages. The goal of this project was to create a completely synthesized model that could accurately represent the flute and be used by others to create music. To create a synthesized flute, physical modeling was used, and Perry Cook's diagram (Fig. 1) and paper were used as references in terms of how the model should be laid out. The choice to use physical modeling was made because it allows for far more efficient usage of memory than FM synthesis or wavetable synthesis, as those require samples to be stored in memory. The use of physical modeling also allows for possible improvements upon past techniques. Before diving into an explanation of how this project's model works, it is important to understand the features of the flute overall.

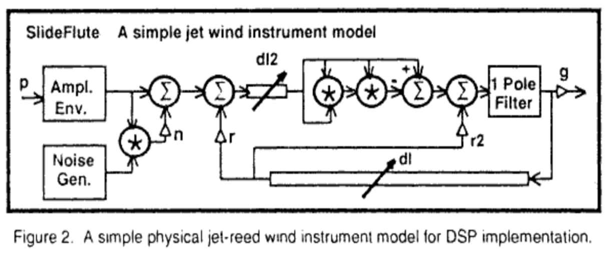
SlideFlute A simple jet wind instrument model
p
Ampl. Env.
n
Noise Gen.
r
dl2
+
-
1 Pole Filter
g
r2
dl
Figure 2. A simple physical jet-reed wind instrument model for DSP implementation.

Fig. 1. Perry Cook's model for a simple synthesized flute. n represents the noise gain with r and r2 as reflection factors, and g and p are the gain and breath pressure, respectively. dl and dl2 are the bore and jet delay lines [5].

II. Background

Wind instruments are some of the more difficult instruments to synthesize without using samples from actual instruments. The flute relies upon the player's airstream and uses the covered and uncovered holes in the body and foot joint of the instrument to determine the notes being played. While other factors, such as the player's embouchure (mouth) shape and the shape of their vocal tract, heavily affect the tone of a physical instrument, the model constructed here is more focused on implementing a flute using a straightforward airstream instead of creating a model of the entire mouth in addition to the flute. With this implementation in place, the initial airstream could then be modified to include the functionality of a player's mouth.

To properly model the inner workings of the flute, jet delay and bore delay were created. The jet delay accounts for the air flowing into the flute from the embouchure hole (the hole that the player blows into) reflecting off the inner wall of the flute before traveling down the pipe. The bore delay accounts for the fact that the flute acts similarly to a pipe where one end is open, and the other is closed. It also accounts for the interaction between the air that is already inside of the flute and the air that enters as a part of the player's airstream [2]. Now that we understand the reasons for including these aspects of the model, we can discuss the components that went into their designs.

III. Design Considerations

A. Breath Pressure

Cook's model of the flute (Fig. 1) uses breath pressure as one of the two parameters required by the model as input, with frequency being the other. Breath pressure is especially important as it determines whether a note fully speaks or not. Without enough pressure, a flute should only have a very faint airy sound, not the full, warm sound associated with good playing. The model accounts for this by scaling the noise gain of the output using the breath pressure. Similarly, when the breath pressure increases above what is needed for a given note, an actual flute will have a grating, overblown sound [2]. In the case of this model, the breath pressure was simply not allowed to reach that point because the desired sound was full and smooth.

B. Attack, Decay, Sustain, and Release

The amplitude envelope referenced by Cook's model corresponds to a simple attack-decay-sustain-release (ADSR) envelope. In an actual flute instrument, the attack is triggered by the player's tongue, and in the model, this type of attack is replicated by a short attack time, with a longer sustain, duration, and release. In a physical flute, each of these parameters can be instantaneously adjusted by the player by changing their embouchure and air stream, but as mentioned earlier, neither this project's implementation nor Cook's model takes this into account.

C. Pitch Creation

A standard flute adjusts its pitch using different fingerings of the keys on the body of the instrument in combination with the breath pressure provided by the player. While Cook accounts for this by adjusting the pitch using jet delay, the model presented here simply uses a sine wave to create the pitch. This was done to get an accurate pitch when selecting the pitch from a MIDI keyboard, as Cook's flute does not always accurately choose a pitch when its jet ratio (a delay factor) is manually changed.

D. Jet Delay

The jet delay of this model uses a standard tapin and tapout processes for the delay, and the delay time changes whenever the user inputs a new frequency. The jet delay takes in the combined signal of the user input and half of the bore delay. This accounts for the reflection and interaction between the bore delay and the airstream mentioned earlier.

E. Bore Delay

The bore delay is slightly more interesting than the jet delay, as it is part of a feedback loop. As seen in Cook's model, the bore delay impacts the flute in two places, just before the jet delay occurs and just after it occurs. The area before the jet delay simulates the reflections caused by the jet and bore airstreams interacting as mentioned earlier. The second reflection occurs at the end of the flute when the airstream hits the end of the pipe [1].

F. Vibrato

While vibrato is not a part of Cook's model for a flute, it is essential to classical flute repertoire and hence was included in this model. The vibrato in this case was created using a sine wave and adding its oscillations to the overall signal of the breath pressure wave. This vibrato changes both the pitch and the volume level of the signal, making it more of a combination of tremolo and vibrato than vibrato on its own. While this implementation may be controversial as this is not true vibrato, the intention is to create a more dynamic performance from the person using the patcher.

IV. Code Implementation

This entire project was made using Max 8. Max was chosen to create a simple user interface, and it also enabled the code to mimic the format of Cook's diagram for easy understanding. Each part of the code was grouped into separate chunks and will now be explained.

A. Breath Pressure and Waveform Creation

To properly form the initial waveform from the breath pressure in Fig. 2, the pressure is initially placed in an ADSR envelope. This converts the float of the breath pressure to an actual signal, which is then given a pitch using the received sine wave. After getting a pitch, the signal adds some white noise to create the breathy quality of the air stream. Finally, vibrato is added, and the signal is sent out to the rest of the patcher. The decision to have the noise signal come before the vibrato is added is what causes the tremolo effect in the vibrato settings.

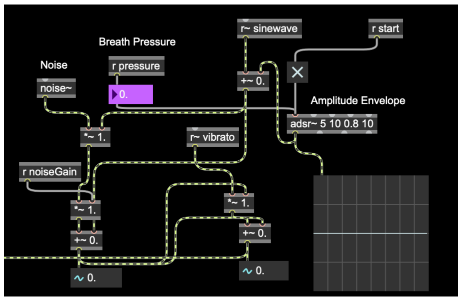

Fig. 2. Input Signal

B. Pitch Creation

The sinewave signal received in Fig. 2 is the result of the pitch creation code shown in Fig. 3 below. The frequency given by the user's MIDI input is given to the phasor and used to create a sine wave. The *~ 6.2813 command simply turns the phasor into radians so that it can be processed by sinx~. After the sine wave has been created, the velocity from the MIDI input is used as the pressure, so the loudness of the pitch scales with how hard the user presses a key on a MIDI keyboard.

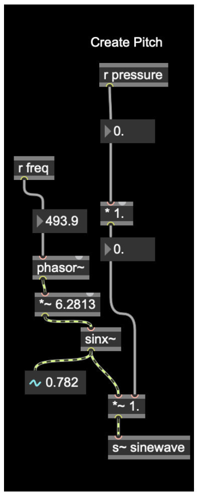

Fig. 3. Pitch Creation

C. MIDI Interaction

As seen in Fig. 4, every time the user presses a key on the MIDI keyboard, the frequency and velocity of the key that is pressed are translated into a note that will sound for 2 seconds. The velocity of that note is then compared to a value of 60 because velocities above that value cause the note to distort and sound less clear. Anything above a velocity of 60 will be changed to have a velocity of 60. After that condition is checked, the velocity is divided by 128 to get a value between zero and one, and then that value is sent out to the rest of the program as the breath pressure. As for the frequency of the note, that is first multiplied by 0.66666 and then divided by the sample rate of the signal to get the number of samples. After that, the number is compared to the length of the bore, and if it is greater than the length, then the length is used in its place. Finally, the quantity is converted to milliseconds and sent off to either the jet delay if it was multiplied by the jet ratio or the bore delay if not. The math done in Fig. 4 to convert the frequency from its initial value in Hertz to its value in milliseconds was brought over from the C++ code that implements Cook's model [7].

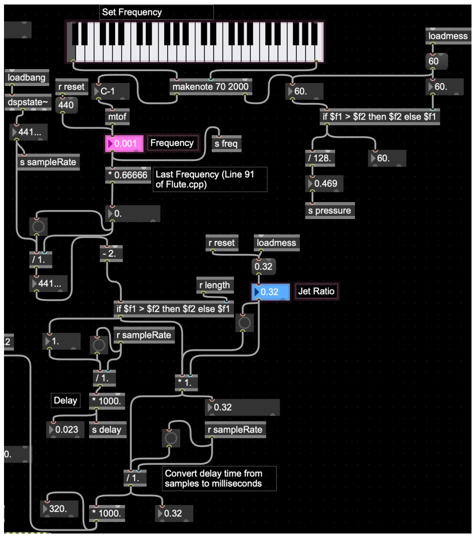

Fig. 4. MIDI Input and Delay Time

D. *Jet Delay*

The jet delay in Fig. 5 is controlled by the tapin~ and tapout~ functions. The amount of time that the signal is delayed comes from the delay time that was found and multiplied by the jet ratio in the MIDI input section of the code. As for the polynomial function implemented after the jet delay, that also comes from Cook's model. The purpose of the polynomial function is to simulate the reflection that occurs at the open end of the bore and its interaction with the incoming air [4].

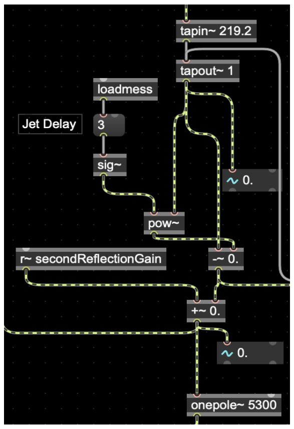

Fig. 5. Jet Delay

E. *Bore Delay*

Like the jet delay, tapin~ and tapout~ control the delay of the signal in Fig. 6, and the delay time comes from the MIDI input section of the code. The signal for tapin~ is received from the output of the lowpass (onepole~) filter, creating a feedback loop within the model. After that signal has been delayed, it goes through either the jet reflection where it interacts with the input air stream, or it goes through the end reflection and is sent to be added to the signal after it passes through the jet delay.

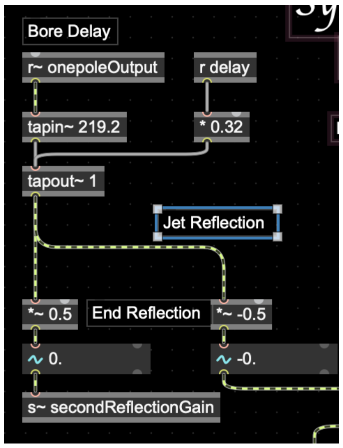

Fig. 6. Bore Delay

F. *Vibrato*

The vibrato implemented in Fig. 7 is very similar to the earlier pitch calculation, but the difference is how the two contribute to the input wave. The pitch is merely added, but the vibrato is both multiplied and added to the signal, meaning that its contour contributes more to the signal than the pitch. As for the initial values chosen for the frequency and gain, those both come from the C++ code used to implement Cook's flute model [7].

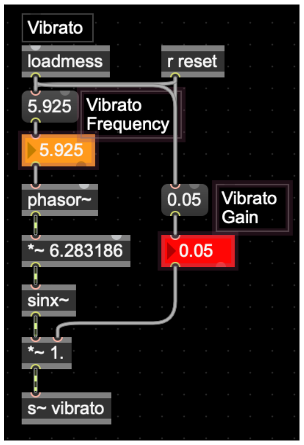

Fig. 7. Vibrato

G. *Lowpass Filter and Output*

The final piece of code is the lowpass filter and the output of the entire model in Fig. 8. The onepole~ filter is a lowpass filter with a pole at 5300Hz, meaning that the lowpass decay starts at 5300Hz. After the signal passes through that filter, it is sent both to the bore delay and the output. This feedback onto the delay means that the signal heard by the listener combines signals occurring at many different times and plays them all at once. This effect is essential to creating the airy sound of the output signal.

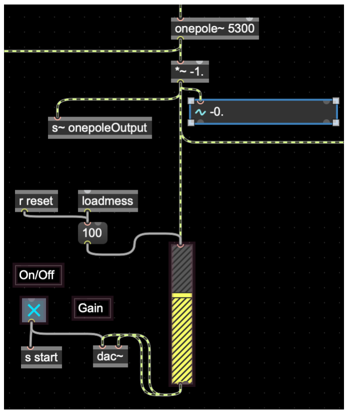

Fig. 8. Lowpass Filter and Output

V. Conclusion

After the creation of the program, spectrograms were used to compare the output of the created model (Fig. 10) with that of Cook's flute model (Fig. 9). Using a spectrogram, one can see that the first four harmonics of the two models are similar, but the model created by this project had only the first four harmonics clearly defined while Cook's model had clearly defined harmonics across many frequencies. Cook's model does have a more flutelike tone than the model created by this project, though it does rely on the user choosing a pressure to use instead of simply using MIDI velocity.

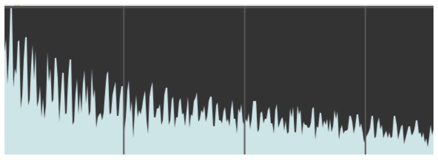

Fig. 9. Cook's Implemented Flute Model

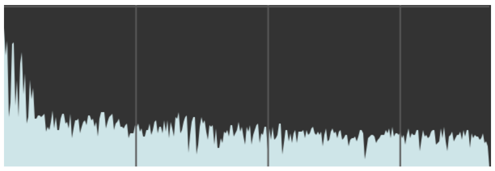

Fig. 10. Synthesized Flute Model

These differences could have occurred due to the way the project's model implemented the jet and bore delays' interactions with the white noise. The jet and bore delays in Cook's model are used to define the pitch of each note, whereas this project's model uses a sine wave to define the pitch.

In the future, this model could be improved upon in several ways. The pitch could be calculated using the jet delay instead of relying upon the pitched sine wave. This would give the airy quality that is expected of the flute instead of a flat tone. Additionally, the mouth of the player could also be modeled to create a more varied air stream. This would give the flute a fuller sound and would provide the user more freedom in adjusting the tone quality of the instrument.

VI. Bibliography

[1] C. Goméz, "MUMT 618 Project Report: Anl Overview of Flute Physical Models," *music.mcgill.ca*, para. 10, December 2018. [Online]. Available: http://www.music.mcgill.ca/~gary/courses/projects/618_2018/Gomez_ProjectReport.pdf. [Accessed Apr. 27, 2022].

[2] G. Ried, "Practical Flute Synthesis," *soundonsound.com*, para. 27, October 2003. [Online]. Available: https://www.soundonsound.com/techniques/practical-flute-synthesis. [Accessed Apr. 27, 2022].

[3] "Max MIDI Tutorial 5: MIDI Advanced Sequencing," *docs.cycling74.com*, para. 5. [Online]. Available: https://docs.cycling74.com/max8/tutorials/midichapter05. [Accessed Apr. 7, 202**2**]

[4] N. Hind, "Physical Modelling Synthesis," *ccrma.stanford.edu*, para. 7. [Online]. Available: https://ccrma.stanford.edu/software/clm/compmus/clm-tutorials/pm.html. [Accessed Apr. 27, 2022].

[5] P. Cook, "A Meta-Wind-Instrument Physical Model and a Meta-Controller for Real Time Performance Control," *quod.lib.umich.edu*, para. 5, 1992. [Online]. Available: https://quod.lib.umich.edu/i/icmc/bbp2372.1992.072/2/-meta-wind-instrument-physical-model-and-a-meta-controller?page=root;size=150;view=image. [Accessed Apr. 27, 2022].

[6] P. Cook and G. Scavone (1995-2002), DelayL.cpp, [C++ Source Code]. Available: https://learn.rochester.edu/webapps/blackboard/content/listContent.jsp?course_id=_68216_1&content_id=_4950637_1. [Accessed Apr. 27, 2022].

[7] P. Cook and G. Scavone (1995-2002), Flute.cpp, [C++ Source Code]. Available: https://learn.rochester.edu/webapps/blackboard/content/listContent.jsp?course_id=_68216_1&content_id=_4950637_1. [Accessed Apr. 27, 2022].