The Birl Physical Model

Application of Real-Time Continuous Tone Hole
Physical Models to the Birl Electronic Wind Instrument

Senior Thesis by
Dharit Tantiviramanond

2015

Submitted to: Princeton University
Department of Computer Science

Advisor

Jeffrey Snyder

https://github.com/reedt/BirlPhysicalModel

This Thesis represents my own work in accordance with University Regulations

Abstract

Physical modeling synthesis is a synthesis technique that relies on the digital simulation of the physical systems embodied in acoustic instruments. Advances in the fields of physical modeling and digital waveguide synthesis with respect to wind instrument modeling made in the past 15 years, particularly tonehole modeling, have provided mechanisms with which accurate and expressive models of wind instruments can be implemented. Specifically, an algorithm to model toneholes that offers continuous control of the tonehole coverage amount via a single parameter has allowed the creation of a wind instrument model capable of reproducing unique acoustic properties that are not possible with other forms of synthesis. Apart from this, the model's physical parameters–tonehole positions, size, and bore size–are also fully customizable. This model, along with the Birl controller, represent a new type of digital wind instrument that offers an unprecedented level of control of expressivity.

Acknowledgements

First and foremost I would like to thank my advisor, Professor Jeffrey Snyder, for being the best advisor I could have asked for. He was a mentor and an inspiration, and I couldn't have complete this project without his guidance and wisdom. He was also just an amazing person to be around, and I feel so lucky to have had the opportunity to see him in action and watch how he goes about "bridging the nerd world and hip world" in a way that no one else does.

I would like to thank Professor Gary Scavone and Professor Perry Cook for their help with the BlowHole STK class and for pointing me in the right directions when I began implementing the project, and also for their *enormous* contributions to the fields of DSP and physical modeling.

I would like to thank Spencer Salazar for being so kind and helping me get the ChuGin framework up and running with the STK. I wouldn't have figured that out on my own. I would like to thank Noah Kaplan for testing out the model and showing me that multiphonics were actually possible with it! I would also like to thank Professor Adam Finkelstein for being my second reader on top of all his other commitments.

I would like to thank my parents and my family for providing me with all the opportunities and resources I ever needed to get to where I am now, but also the wisdom, companionship, and freedom that shaped me into who I am today. Even though we're across the world from each other, I'm forever grateful for your love and support. None of this would have been possible without you guys.

I would like to thank Riley, Scott, Louis, and Greg for helping me talk through some of the concepts in the project that were stressing me out in the times when i was stuck.

I would like to thank the OG Forbes Hoodrats. We've been there for each other since the beginning and we will be here for each other until the end. And I would like to thank Terrace, for teaching me to make my fantasy a reality, and to live it with abandon. I will miss you all dearly once our time is up.

1. Introduction

In real wind instruments, toneholes provide a wide range of spectral and tonal possibilities through the variety of ways that a performer can interact with them, but much of this flexibility is lost in these instruments' electronic counterparts. Completely covering a tonehole effectively shortens the length of the instrument bore, dropping the pitch to a lower frequency, while opening a tonehole does the opposite. Covering it a fraction of the surface area interpolates between these two pitch boundaries. And a combination of higher breath pressure and certain coverage values can produce unexpected acoustic effects such as multiphonics, the production of more than one tone at once. These techniques are all possible due to the physical characteristics of the tube and the unique way that air pressure waves blown into it interact with the bore cavity, and how toneholes dynamically alter these properties.

Current electronic wind instrument controllers offered on the market, such as the Akai EWI models and the Yamaha WX series, do not provide this type of flexibility. Their buttons and capacitive sensors can only be either fully open or closed, and as such, their range of expressiveness is severely limited when compared to actual wind instruments. Not only that, but most existing digital physical modeling synthesizers are built to accept MIDI input so that they can operate within modern DAW (digital audio workstation) environments such as Ableton or Logic. These restrictions limit the sonic and physical possibilities to anyone seeking to extend an electronic music piece with a component from the wind instrument realm. This project, in combination with the Birl project, seeks to push these limitations by combining recent advancements in the field of physical modeling and digital waveguide synthesis with a new type of physical controller capable of continuous parameter input at the toneholes. The software developed as part of this thesis is aimed towards providing a match for the Birl in terms of flexibility and similarity to acoustic wind instruments and to unlock new potential in digital synthesis that was previously unavailable.

More specifically, the goals of this project are to build a model of a wind instrument that is capable of accurately replicating certain acoustic effects that are unique to physical wind instruments, and

can also be customized in ways that are impossible with real instruments. In this way, the model will represent a step forward in wind instrument physical modeling synthesis, and will also offer important advantages over the physical instruments that it seeks to simulate. The key contributions of this project are manifested by features of the implemented model:

- Customizability at the level of number of toneholes, tonehole radii and position, and bore radius.
- A tuning interface that allows users to choose from a set of pre-defined tuning systems, or specify a custom tuning by defining a set of desired frequencies in a scale in relation to a fundamental pitch.
- Capability to produce sonic effects unique to wind instruments such as overblowing and multiphonics.

However, certain shortcomings of the final model must also be mentioned. The most egregious is a noticeable discrepancy between the desired frequencies that the user specifies and the actual frequencies output by the model. This problem implies that the model is not tuned absolutely–it is only tuned relative to itself–and will not sound in-tune with other instruments in its current state.

A survey of the background material and research that this project is founded upon is offered in section 2, followed by the implementation details and contributions of this project in section 3. An evaluation of the final product can be found in section 4, and some directions for further work relating to this project are laid out in section 5. Finally, the report's conclusion is in section 6.

All code associated with this project, as well as recordings of the instrument in action, can be found online at `https://github.com/reedt/BirlPhysicalModel`.

2. Background and Related Research

An introduction to the overarching Birl project, of which the Birl physical model implemented in this project is but one part of, is given in section 2.1. A survey of the knowledge and prior research in the field of physical modeling synthesis necessary for the completion of this project is offered in section 2.2, followed by an examination of the mathematics behind tuning a simple wind instrument in section 2.3.

2.1. The Birl

The Birl project began in 2009 as part of Professor Jeffrey Snyder's doctoral dissertation [20]. The goal of the project is to produce a high-quality alternative to electronic wind instruments currently offered on the market, with an emphasis on the expressiveness of the synthesis controls and customizability. The Birl's physical body has already been constructed by Professor Snyder, and runs on three STM32 ARM 32-bit microcontrollers operating at 168 MHz. Its components include a breath pressure sensor and capacitive sensing brass tone hole screws. Some new models also include extra buttons and two capacitive sensing XY pads for input of other parameters. The toneholes produce continuous input that corresponds to the amount of surface area coverage by the user at any given moment.

Figure 1: The Birl electronic wind instrument.

Currently, the Birl functions as a controller that transmits MIDI and/or OSC (Open Sound Control) messages corresponding to user input on to a computer. A long-term goal is for the Birl to be a standalone instrument that synthesizes sound output directly without having to be connected to an external machine. Synthesis is already possible in the Birl, but no current digital synthesis software is built to handle

input in a way that is compatible with the Birl's architecture. Specifically, the Birl sends tonehole coverage data in a way that exactly replicates a wind instrument–that is, each tonehole is given its own parameter–but most synthesizers are configured to accept input that corresponds directly to the desired pitch to produce. To handle the Birl's output in an intuitive way, a synthesizer would have to process the tonehole data in some way to determine what resulting pitch to output.

Three possible approaches have been conceived as intuitive and acceptable forms of synthesis and interaction for the Birl:

- Machine learning/neural net-based approach: A neural net trained on user input data in the form of mappings from tone hole fingerings to desired pitch output could theoretically produce intuitive behavior given enough training data, and can even be customized to suit the specific needs of a user. Professor Snyder implemented this approach, and found that although usage is generally intuitive, sometimes the neural net will perform unexpected interpolations that lead to undesired sound artifacts. Also, the training phase is somewhat complicated, as the user must input a large number of fingerings and specify the exact corresponding desired sound [11].
- Rule-based approach [1]: A system of rules specifying mappings between fingering configurations and pitch output was implemented which could yield pitches for most fingerings by interpolating between the pitches for a set of known fingerings. To account for unconventional fingerings, a web-scraping service was also implemented that generates a large lookup table to assign pitches to all possible fingerings from an online resource. This approach is simple and efficient, and works very well for the specific pitch system built into the code. However, it is not extensible by the user and cannot be easily adapted to suit other tuning systems. It also does not account for the acoustic physical properties unique to wind instruments such as overblowing.
- Physical modeling approach: This involves implementing a model that uses digital audio processing techniques to simulate the physical characteristics and behavior of an acoustic tube that resonates when air pressure is input into the system. The physical modeling approach is the central focus of this project, and a discussion of the engineering of such a model follows in section 2.2.

2.2. Physical Modeling and Digital Waveguide Synthesis

Physical modeling synthesis is the process of using digital models of physical systems to generate sound. By implementing a mathematical model that simulates the ways in which a physical instrument reacts to various forms of input, as well as its surrounding environment, oscillations in various materials involved in the model can be sampled to produce sonic output. The authenticity of the sound is directly related to the quality of the model's replication of its corresponding physical system.

The particular type of model implemented as part of this project is known as a digital waveguide model. A *digital waveguide* is in essence a delay line, and digital waveguide models use delay lines extensively to model the propagation of signals such as position displacement or air pressure through materials like a string or a tube. Delay lines are useful components in these models because they can easily and efficiently simulate the movement of a signal through an object, and varying the delay line length is equivalent to varying a size parameter of the object [9].

The physical modeling background necessary to understand this project is easily achieved through the explanation of the fundamental building blocks that make up the system. We will first discuss the Karplus-Strong algorithm, one of the most basic algorithms that is the at the heart of many different digital waveguide models. Then, we will describe how different generalizations and extensions can be applied to the Karplus-Strong model in order to obtain the final model used in this project. These extensions include some recent advancements in physical modeling synthesis that provide the aforementioned continuous tonehole control, one of the key contributions of the project overall.

2.2.1. Karplus-Strong algorithm The Karplus-Strong algorithm models the displacement of a plucked string over time [9]. To understand the algorithm, consider the behavior of a taut string connected at the right and left ends to rigid terminations. When plucked, the string oscillates upwards and downwards as the displacement moves along the length of the string. This pattern of movement is known as *transverse waves*, as the direction of oscillation is perpendicular to the direction of energy transfer. Notice that the waves are propagated in both the left-going and right-going directions. At the terminations, the displacement is reversed: positive displacement

becomes negative displacement, and vice versa. The superposition of the two waves traveling in opposite directions produces a phenomenon known as *standing waves*. At some points along the string, the two waves' amplitudes will always cancel out for the duration of the oscillation. Such points are known as *nodes*. And at other points (in between nodes), the amplitudes will sum to be double the amplitudes of either traveling wave, resulting in an oscillation between this positive and negative sum value. These points are known as *antinodes*. Furthermore, the displacement decreases with time as there is a small energy loss due to friction with surrounding air and other factors. Contact with either of the terminations also results in a more substantial energy loss.

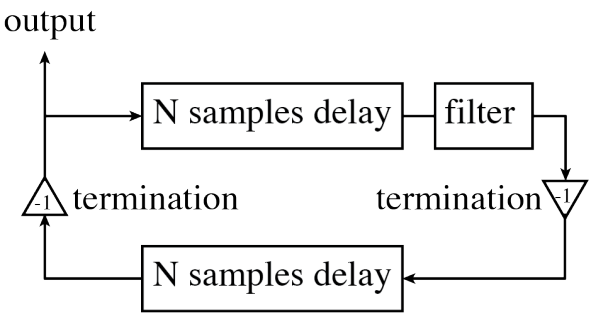

Figure 2: The bidirectional Karplus-Strong algorithm [9].

This system can be modeled with digital waveguides as shown in Figure 2 (content taken from [9]). Two delay lines are used to model the left-going and right-going directions of displacement propagation. Behavior at the terminations is modeled with a phase inversion (multiplication by -1). Loss due to friction with air and the terminations can be lumped into a single filtering operation (because the loss operations are associative). A averaging lowpass filter is a sufficient approximation of these loss operations for a simple model. The action of plucking a string is modeled by initializing the delay line buffers with random noise. Then, the simulation can begin by running the following loop for each time step:

1. One sample is drawn from the right-going delay line, passed through the lowpass filter, and its phase is inverted. The result is input into the left-going delay line.
2. One sample is drawn from the left-going delay line, and phase-inverted as well. This sample is input into the right-going delay line.
3. An output sample is constructed by taking the sum of two samples at a particular location along the two delay lines.

In this way, the system forms a continuous cycle that models the oscillation of position displacement as time progresses. The amount of displacement decays with each time step, as the string will eventually stop moving. The rate of decay depends on the amount of filtering applied.

Readers familiar with the Karplus-Strong algorithm may notice that its presentation here deviates from the form that it is usually introduced in, which comprises only a single delay line. This optimized model is derived from the model shown here by noticing that *all* the losses can be lumped into a single calculation, again due to associativity, and the fact that we sample output at only one position. This implies that the two delay lines may be combined into one delay line of double length. The two inversions will also cancel out as a result. For the purposes of this project, it is simpler to introduce the algorithm in this unoptimized form, as this is more similar to the model implemented in our prototype. [9].

2.2.2. Wind Instrument Extensions to the Karplus-Strong Algorithm In a wind instrument, air pressure waves are generated in the instrument bore by blowing into an embouchure hole or a mouthpiece with a reed attached to it. With an embouchure hole, the player's breath stream is split by the edge of the hole, sending the stream alternately into and out of the bore. This is the source of the oscillations in air pressure. With a reed instrument, the reed acts as a gate for the incoming air stream. The vibration of the reed opens and closes the entryway at the mouthpiece, which forms the air pressure waves inside the bore[1] [10].

These air pressure waves are known as *longitudinal waves* (see Figure 3), as the direction of displacement oscillation is parallel to the direction of energy transfer. The propagation of these waves follows the same patterns as the movement of transverse waves in a plucked string, and therefore the Karplus-Strong algorithm can be extended to model an acoustic tube as well. Some modifications must be

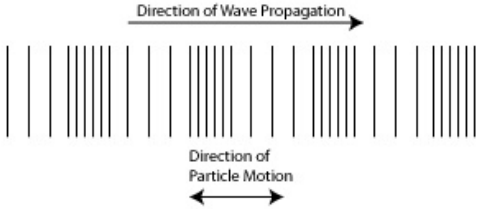

Figure 3: A visualization of the longitudinal air pressure waves inside an air column.

[1] In fact, the reed also vibrates with the air pressure waves inside the bore. The two are interdependent processes.

performed to account for key differences between the two systems, however. Most importantly, a model for the losses incurred by toneholes must be added. A discussion of such models is offered in the following section, but some other minor modifications are considered here first.

Again, two delay lines will form the models for the bidirectional propagation of the signal (from mouthpiece to bell[2] and from bell to mouthpiece). Breath pressure values continuously input into the system to mimic the way that a player's breath enters an acoustic tube, with added random noise to account for turbulence. For a reed instrument, the reed's effects on the signal can be simulated by manipulating the reflection coefficient at the mouthpiece end of the model. The exact value of the coefficient follows a nonlinear function that is parametrized by the incoming reflected signal as well as incoming breath pressure. The difference between these two signals is known as the *pressure differential* and forms the lookup index to this function. Such a function can be sampled and loaded into a *reed table* for various values of the differential. The reed's physical mass is small enough relative to the other components to be omitted from the model [7].

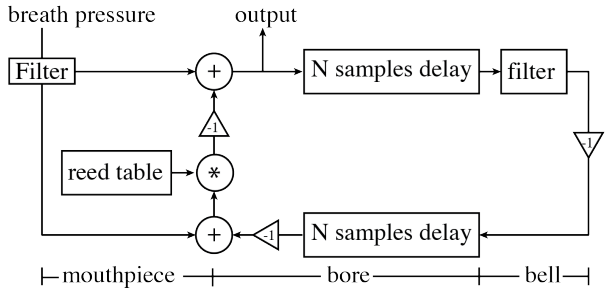

Figure 4: A basic schematic for a wind instrument physical model based on the Karplus-Strong algorithm [7].

Another important difference between the string and tube Karplus-Strong models is the termination reflections. At the end of a tube, phase inversion occurs only if it is an open end. For a closed end, the pressure signal is reflected without a phase inversion [2]. For most wind instruments, therefore, the signal's phase should be inverted only at the bell end, and not at the mouthpiece end.

A basic schematic detailing the Karplus-Strong algorithm with these added modifications is provided in figure 4 (content taken from [7]).

2.2.3. Tonehole Physical Models [17], [3], and [4] offer developments in efficient tonehole modeling by detailing the behavior of scattering at the tonehole position in the bore, as well as the

[2]We use the term *bell* to mean the open end of the wind instrument opposite the mouthpiece, according to clarinet terminology.$^{food=love}$

signal loss effects caused by the tonehole directly. These papers contain the key ingredients that make this project possible–prior to these, no method was available for modeling toneholes such that they could be controlled with a single parameter.

At the three-way junction between the tube bore and the tonehole branch, a model for the scattering behavior of the air pressure waves arriving from the three different directions is proposed according to figure 5 (content taken from [4]).

In the figure, the important input and output variables are labeled in the according manner:

- Pressure variables are first labeled according to their position in relation to the junction. So, the pressure variables related to the section of the bore appearing before the junction are labeled pa, variables related to the bore section appearing after the junction are labeled pb, and variables related to the tonehole branch are labeled pth.
- The variables are superscripted with their direction of travel. Movement away from the mouthpiece end (the source of the breath pressure fueling the waves) is considered the positive direction. Therefore, pa^+ denotes the air pressure value coming from tube section A going into the junction, while pa^- denotes the air pressure leaving the junction and moving towards the mouthpiece as a result of the scattering operation. Similarly, pb^+ and pb^- are the pressure values moving into tube section B and coming out of B, respectively, and pth^+ and pth^- are the pressure values moving into and out of the tonehole branch, respectively.

The r_0 variable is known as the scattering coefficient. Its value depends on the characteristic admittances or impedances of the bore and the tonehole branch, which in turn depend on the volume of airflow at a given moment as well as the radii of the bore and tonehole branch respectively. The equations relating this relationship, as well as the equations involved in the scattering junction, are listed explicitly in [17].

The second component in tonehole filtering is a model of the loss effects of the tonehole on the incoming pth^+ signal. According to [4] and [3], these effects can be simulated with an allpass filter with a single coefficient whose value is dependent on the characteristic impedances of the bore and the tonehole branch. These values can be calculated from the respective effective radii of the

corresponding tube sections. The explicit discrete-time equations are presented in [4].

An attempt at combining these models with a controller similar to the Birl is mentioned at the end of [4], but the results are not discussed in detail and the prototype does not seem to be available.

2.2.4. STK Much of the aforementioned functionality is freely available as part of Synthesis Toolkit library (STK) implemented by Perry Cook and Gary Scavone. The library is written in C++ and includes implementations of various digital signal processing algorithms and techniques. In particular, it includes the BlowHole class which implements the tonehole models outlined in section 2.2.3.

The main body of the model is made up of the single delay line version of the Karplus-Strong algorithm, outfitted for the purposes of modeling an acoustic tube through the addition of a reed table and other appropriate modifications. However, BlowHole only includes a single tonehole, and is engineered to accept MIDI input. The frequency corresponding to the MIDI note is synthesized by changing the length of the bore of the model such that it will be tuned to the desired frequency. This form of interaction makes BlowHole an excellent option for various use cases such as as part of the arrangement of an electronic music composition. Since the model only includes one tonehole, it is small enough that its behavior is reliable, predictable, and accurate. As a live instrument, however, it would only be capable of doing the same thing: producing pitches according to MIDI note requests. Acoustic effects unique to wind instruments are lost, since it does not accurately replicate the physical characteristics of an actual wind instrument. It also does not reproduce the way a performer interacts with the instrument, as it lacks a method of translation between fingering configurations for multiple toneholes and sound output.

Implementations of the reed table function, as well as other various DSP (digital signal processing) building blocks such as filters and delay lines are also implemented in the STK. We relied heavily on these STK classes during the early phases of the development process.

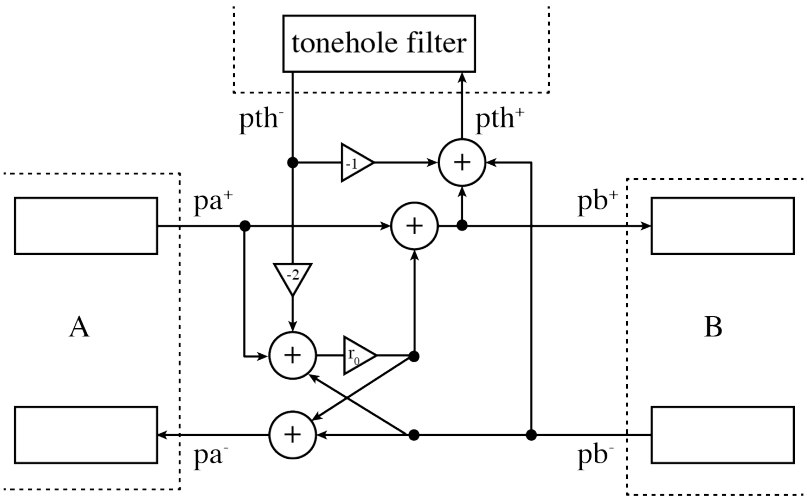

Figure 5: Air pressure scattering at the tonehole location [17] [4].

2.3. Wind Instrument Tuning

A discussion of the implementation of the tuning module that is one of the contributions of this project requires some background knowledge relating to the mathematics behind wind instrument tuning. This section derives largely from [2], and a more in-depth discussion of these concepts can be found there.

The wavelength of a signal is defined as the total physical distance it travels before it repeats itself. The *fundamental frequency* of a tube is the lowest frequency at which air pressure waves can oscillate inside the tube. To understand how the wavelength of the fundamental is calculated, imagine an *open-open* tube open at both ends, with the two ends labeled A and B. Let us assume that no energy will be lost due to friction or other factors. Now imagine a point of compression entering a as part of an air pressure wave at end A. An air pressure wave is inverted upon contact with either end of this tube, so when that point of compression reaches end B it is inverted into a point of rarefaction and begins traveling back towards A. Once it reaches A, it inverts into a point of compression again, and thus we know that this is the point at which the oscillation pattern repeats itself. The point of compression traveled the length of the tube twice before repeating itself, and therefore the wavelength of the fundamental mode is twice the length of the tube.

For a *closed-open tube* that is closed at one end and open at the other, the same thought experiment

can be applied to find the fundamental wavelength. The imagined point of compression will travel the length of the tube four times before becoming a point of compression again (since the signal will not become inverted upon reaching the closed end), and therefore the fundamental wavelength is four times the length of the tube. Applying this equation to the equation relating wavelength and frequency results in

$$\begin{aligned} \lambda &= 4 * L_T \\ F_1 &= \frac{c}{\lambda} \\ F_1 &= \frac{c}{4L_T} \end{aligned} \tag{1}$$

where λ is the fundamental wavelength, L_T is the tube length, F_1 is the fundamental frequency, and c is the speed of sound.

In actual wind instruments, however, air pressure waves actually travel a small distance beyond the open end of the tube. Therefore, the fundamental wavelength will actually be slightly longer and the fundamental frequency slightly lower than what would be expected from the calculations using (1). This implies a discrepancy between the *effective length* or *acoustic length* of the tube (which corresponds to L above) and the *cut length* of the tube, which refers to the actual length. We call this difference the *end correction* of the tube and label it Δl_T.

The intuition is that for a closed-closed tube, $\Delta l_T = 0$, since the pressure waves do reflect exactly at the ends of the tube. A closed-open tube of cut length l_T will produce a fundamental frequency equal to that produced by a closed-closed tube of the corresponding effective length L_S. We refer to this fictitious tube as the *substitution tube*. The relationship is given by $L_S = l_T + \Delta l$. Forster provides this equation for estimating Δl:

$$\Delta l_T = 0.3 d_1 \tag{2}$$

where d_1 is the bore diameter.

This phenomenon applies to the length of the tube at each tonehole as well. We refer to the length

from the mouthpiece end of the tube to the center of any given tonehole i as l_H, which is equal to the effective length of the tube when all toneholes before tonehole i are closed. This action is equivalent to shortening the cut length of the tube to a value denoted by l_H, which has a corresponding effective length $L_{S(h)}$. The calculations for the tonehole correction are given by:

$$\Delta l_H = z L_{S(h)} \tag{3}$$

$$z = 0.5g\sqrt{1+4\frac{L_{B(h)}}{gL_S}} - 0.5g \tag{4}$$

$$L_{B(h)} = (l_H + d_H)(\frac{d_1}{d_H})^2 - 0.45d_1 \tag{5}$$

d_H refers to the tonehole diameter, and $L_{B(h)}$ is a variable that does not have a physical manifestation, but refers to "the length of a fictitious duct that has the same diameter as the main tube, and the same conductivity as the tone hole" [2]. g is the interval ratio between two toneholes on the tube, and therefore the exact value depends on the tuning system in use. With just intonation, $g = \sqrt[12]{2}$, while for an equal tempered scale $g = r_i \div r_j$, with r_i and r_j being the interval ratios of toneholes i and j with respect to the fundamental.

To solve for the position and radius of a tonehole, (3), (4), and (5) must be combined with (1) to find suitable values for l_H and d_H. First, the desired pitch output must be chosen, from which $L_{S(h)}$ can be determined using (1). Then, a reasonable starting value must be chosen for either d_H or l_H which can be used to solve (3).

3. Prototype Implementation

With the background in mind, we move on to a discussion of the implementation of the Birl physical model prototype that was built for this project. The development environment and tools involved are examined in section 3.1, and design decisions made, key contributions of the project, and some interesting obstacles encountered over the course of the project are outlined in section 3.2.

3.1. Development Environment

The physical model was implemented as an external "ChuGin" that could be installed and included in ChucK shreds. Other tools such as Max/MSP, Jack, and DAWs such as Ableton or Bitwig were also useful for certain parts of the process.

3.1.1. ChucK and ChuGins ChucK is an open-source digital signal processing and synthesis environment [5]. It provides a "strongly-timed" model that allows for fine-grained control of event sequencing as well as event-driven reactive programming. This, combined with its concurrency model, encourages users to define small units of functionality that can be executed as parallel threads, then add them all to the ChucK virtual machine for combined effects. It also provides a useful interface to interact with MIDI and OSC message data. ChucK is distributed with MiniAudicle, an integrated development environment that provides features such as syntax highlighting and shortcuts that simplify many ChucK related tasks.

ChucK's synthesizers are known as "ugens," or unit generators. All of the STK classes are ported as built-in ugens. Recently, ChucK has also grown to support a new framework for adding external ugens and effects known as the "ChuGin" framework [6]. The framework allows developers to code signal processing objects in C++ and define APIs through which users can interface with their classes. The main execution loop is defined in a function named `tick()`, which is called by ChucK once every sample step.

The recentness of these advances are exciting, but also mean that the framework is somewhat undocumented and untested. One of the earliest stumbling blocks in this project simply involved getting the ChuGin framework set up correctly and coercing it to successfully compile included STK classes. The ChucK source code, the ChuGin source code, and the STK source code can downloaded from [5], [16], and [14]. The necessary set up steps for a Macbook Pro running OS X 10.9.5 are laid out here for the purpose of facilitating the recreation of the development environment described in this report:

- It was necessary to re-copy the ChucK header files (`chuck-[version]/src/*.h`) into the directory `chugins/chuck/include`.

- A new ChuGin can be created by running chuginate/chuginate from within the `chugins` directory.
- The STK source code must be copied in its entirety to the newly created ChuGin's directory.
- Line 5 of `makefile.osx` must be changed to be `FLAGS+=-D__MACOSX_CORE__ -I$(CK_SRC_PATH) $(ARCHOPTS)`.
- `FLAGS+=-Istk-[version]/include` must be added to the first line of `makefile`, and the version number must be changed to match the STK version in use.
- STK classes in use must be added to the `CXX_MODULES` variable in `makefile`. For example, a ChuGin named `dummy` that makes use of the STK version 4.5.0 Noise class would set the variable as such: `CXX_MODULES=dummy.cpp stk-4.5.0/src/Stk.cpp stk-4.5.0/src/Noise.cpp`.

3.1.2. Other tools Max/MSP is a proprietary DSP and musical performance environment. It shares a common ancestry with Pure Data, its open-source counterpart, and boasts a vibrant user community as well as growing support for graphics processing as well. It is an excellent environment for quick tests and musical calculations, and Max includes demonstrable examples of many useful DSP objects, such as filters. As a result, we used Max both as a learning tool and a music-related calculator. We routed audio from MiniAudicle to Max using the open-source audio routing program Jack.

- We used the `filtergraph~` object to simplify the filter coefficient calculation process.
- The `fiddle~` object performs sinusoidal decomposition on an input signal and can be useful in determining the fundamental frequency and harmonics output by the model.
- The incoming audio signal's frequency response can be viewed under a 3rd-party equalizer plugin using the `vst~` object. This task can be performed by routing the signal through a track in a DAW as well.

3.2. Implementation Details

The wind instrument physical model implemented in this project models an acoustic tube with a reed at the mouthpiece, a variable number of toneholes along the bore, and a constant bore radius. With these specifications, its closest physical counterpart is a clarinet. This section describes

the evolution of the model up to its current state. Section 3.2.1 starts with its foundation as the BlowHole STK class . Then, some important technical considerations that arose due to the extensive use of digital waveguides in the model are examined in sections 3.2.2 and 3.2.3. A discussion of the implementation of the tuning system is offered in section 3.2.4, followed by a listing of additional signal processing features that were added to the model's signal chain in section 3.2.5. Finally, documentation for using the ChuGin in ChucK is given in section 3.2.6.

3.2.1. BlowHole as a Foundation Although the BlowHole STK class was implemented for purposes much different than the goals of this project, it was still incredibly useful as a starting point. Much of the basic functionality of an acoustic tube model was already in place and guaranteed to work, which made starting the project a much less daunting task. In particular, the model was set up with a functioning reed table and correct signal transfer schemes at the terminating ends. Furthermore, the tonehole model was already set up as a pole-zero filter, and the equations relating the coefficients and the tonehole radius were implemented as part of the tonehole initialization according to [17] and [4]. An accessor method that modified the coefficients in the correct manner given a tonehole coverage value was also present.

One part of the model that we were unable to use was the main delay line body of the model. The authors of the class performed optimizations that were not outlined in [4] which condensed the two delay lines and the loss and scattering functions into a single delay line for efficiency purposes. This model was more difficult to understand as many of the operations were lumped together, and it lacked documentation pertaining to how the optimizations were calculated. After many unsuccessful attempts, we were unable to extend this model to support further toneholes. Therefore, we re-implemented the body of the model as a dual-delay line bidirectional version that more closely resembled the schematic outlined in [4] and figure 4.

The model is set up as thus: the body is divided into segments, and each segment is made up of two delay lines, each modeling an opposite direction of travel. There is a segment between the mouthpiece end and the first tonehole, a segment dividing each tonehole, and a final segment appearing after the last tonehole. The lengths and the radii of the toneholes are variable according

to the tuning chosen, as well as the bore radius. The vent was omitted because we were unable to translate the air pressure scattering behavior at the vent junction in the single-delay line model of BlowHole into a dual-delay line equivalent, and it was determined to be of secondary importance to the toneholes.

At the mouthpiece end, the algorithm describing the reactions between the incoming breath pressure and the reflected pressure signal was drawn from BlowHole. A lowpass filter and reflection coefficient at the bell end was set up in a manner similar to BlowHole's implementation as well. At the junction of each tonehole branch, the scattering function was implemented to follow [4] exactly. A step-through of the operations contained in the ChuGin's `tick()` function and performed at each sample step is given here:

- The pressure differential between the new breath pressure and reflected air pressure from the body is calculated and used as a lookup parameter to the reed table. This yields a value that forms the input to the model body.
- For each tonehole, the scattering function calculations are performed using pa^+, pb^-, and pth^-, yielding new values for pa^-, pb^+, and pth^+ according to the schematic in figure 4. These values are saved in buffers to be input into the delay lines later.
- A bell-reflected value is calculated by taking the signal value from the delay line at the bell end and passing it through a lowpass filter, reflecting it, and decaying it by a small amount.
- After all the calculations have been performed, the pressure values stored in buffers are input to their respective destinations in a single update phase.

Notice that instead of taking the results of the scattering function calculations and inputting them into their destination filters or delay lines immediately, the values are stored in buffers for use later. They are input to their destinations all at the same time in an update phase. This is because if input operations were interleaved with calculations, the model's state would be inconsistent between calculations. Imagine performing the scattering function calculations for two toneholes i and j, with i and j being connected by a tube segment. This segment would contain two delay lines, one of which transfers the signal from i to j. Let us label this delay line u. If we first perform the

scattering calculations for i at time step t, then immediately take the results and plug the correct value of pa^- into u, this results in an inconsistent state. This is because u would now contain values corresponding to the signal at time step $t+1$. To understand why, contrast this scenario with the ideal behavior in a real wind instrument. At time step t, the scattering at i and j will occur *at the same time*. The scattering occurs by incorporating signal values from the surrounding tube segments, which are all consistent with each other for time t. By taking pa^- and pushing it into u before the scattering at j, we have effectively put u into the same state it would be in at time $t+1$ before the scattering at j for time t has had a chance to process.

This separation of the calculation and update phases is an important concept in physical waveguide synthesis, and is one that is also easy to overlook. One of the major roadblocks in the development of the model was a bug caused by interleaving the two phases.

3.2.2. Integer-Length Delay Lines Broadly speaking, there are two types of digital delay line implementations: integer-length delay lines and fractional delay lines. Integer-length delay lines are delay lines with whole number lengths. As a result, they can only output signals delayed by a whole number of time steps. Fractional delay lines are implemented in nearly the same manner as integer-length delay lines, but with the additional functionality of being able to output signals delayed by a fractional amount of time steps instead of being constricted to whole numbers. The way this is achieved is by using interpolation between the integer time steps. For example, if a signal delayed by 7.5 time steps is desired, a fractional delay line will interpolate between the delay values stored for time steps 7 and 8 to give an approximation of the delayed signal at 7.5.

Although fractional delay lines can seem much more flexible at first glance, there is a tradeoff associate with their use. Because interpolation is employed to compute the fractional delays, the quality of the output signal depends on the quality of the interpolation function. Even with a high-quality interpolation function, there will always be a small error discrepancy between the interpolated value and the actual value of the signal at that fractional time step. This error is insignificant enough to be acceptable for most use cases, but can propagate and accumulate in a system where multiple delay lines are chained together such that signals are passed from one to the

other. In this situation, the error can manifest itself as distortion of the signal. Unfortunately, this is exactly the system that forms the body of the model in this project.

In order to avoid distortion, a design decision was made early on to minimize the use of fractional delay lines in the Birl physical model. This decision was supported by the fact that the tuning equations described in [2] are parametrized by two parameters: the tube length l_H and the tonehole radius d_H. This means that they are set up to compute a good value for d_H that corresponds to a given value of l_H, or vice versa (computing l_H for a given d_H). This implies that there are many possible solutions to the equation system, and therefore a correct value of d_H exists for any integer value of l_H. Fractional values of l_H are unnecessary and can be avoided.

One consequence of this decision is if the spacing between two desired pitches for two adjacent toneholes is small enough, the corresponding integer lengths for the delay lines at the two toneholes will be the same. In other words, the integer lengths of the tube at these two tonehole locations clash, and the result is that one of the tube segments will be given an integer length of 0. A clash will occur if the tonal distance between the two toneholes corresponds to a delay line length of less than or equal to one sample. The equation to determine whether a clash will occur is given here:

$$\left| \frac{F_S}{4f_1} - \frac{F_S}{4f_2} \right| < 1 \tag{6}$$

where F_S is the sampling rate, and f_1 and f_2 are the desired frequencies at the two toneholes. This equation represents first converting the two frequencies to the corresponding delay line lengths that will produce those frequencies, then checking to see if the difference of these lengths is less than one sample. If this is true, a clash can possibly occur but is not guaranteed to occur. If this is false, however, a clash is guaranteed not to occur.

3.2.3. Oversampling Another approach to solving the issue of delay line length clashes is to increase the resolution of the system to increase the space of possible integer length values that the delay lines can take on. This is achieved in the model by a technique known as *oversampling*, which involves increasing the sampling rate of the system. To understand this concept, recall that a

ChuGin's `tick()` function is called once every time step. The exact rate at which `tick()` is called is equal to the *sampling rate*, which is the speed at which the audio engine processes and outputs audio samples. In most modern digital audio architectures sampling rates of 44,100 Hz or 48,000 Hz are most common, and in the ChuGin framework and STK library the sampling rate is set to 44,100 Hz.

Oversampling, therefore, involves artificially increasing the sampling rate of the prototype internally by increasing the number of computation loops performed in the model per call to `tick()`. In other words, oversampling is accomplished by running the model's simulation more than once per time step. However, the system's interaction with the external audio engine does not change. Since we cannot increase the number of input breath pressure values we receive per time step, interpolation is performed on the input breath pressure signal to generate more fine-grained input values, and the output samples are averaged so that the model still only outputs one value per time step to the audio engine.

A desirable side effect of oversampling is that the resolution of the system must be proportionally increased in order to maintain equivalence with the model prior to oversampling. Consider two delay lines: delay line A is in a non-oversampled system, and delay line B is part of a system that is being oversampled by a multiplier of 2. If both have the same lengths, a sample input into B will travel through B in half the time it takes to traverse A. To correct for this time difference, the length of B can be multiplied by the oversampling multiplier. Then, input samples will traverse through A and B in the same amount of time. Since B's model simulation is running twice as fast, B's length must correspondingly be twice as long in order to produce the same pitch as A.

The effects of oversampling can also be understood by examining equation 6. According to the equation, oversampling reduces the maximum tonal distance between the desired pitches of two adjacent toneholes at which an integer-length clash will occur.

An unexpected consequence of oversampling that was observed in the Birl model was a detuning effect that increased with the oversampling multiplier. The effect was audibly noticeable with a multiplier of 4. The exact cause of this artifact is currently unknown, but a multiplier of 2 was

chosen as currently optimal for this project in order to avoid detuning.

3.2.4. Tuning System The tuning system of the model is a module that encompasses the equations outlined in section 2.3. Tuning of the model is achieved by setting appropriate tube cut length l_T, bore radius d_1, and tonehole radii d_H parameters given a set of desired frequency outputs for the toneholes. Intermediate steps involve the length of the imaginary corresponding substitution tube L_S or $L_{S(h)}$.

An outline of the basic algorithm for tuning the main bore given the fundamental frequency runs as thus:

1. Compute L_S from the desired fundamental frequency using (1).
2. Compute a suitable integer value for l_T using $\lfloor L_S \rfloor$.
3. Compute the corresponding value for d_1.
4. If the value of d_1 is less than some pre-defined minimum threshold value, subtract 1 from l_T and repeat steps 2-4.

The resulting values of d_1 and Δl_T are suitable parameters to tune the tube's fundamental frequency. The same basic idea is used to find tuned parameters at the toneholes, but with the equations that relate d_H and $L_{S(h)}$ instead. Algebraic manipulation of the equations that appear in [2] yields these equations:

$$L_{B(h)} = \frac{(L_{(h)} + 0.5gL_{S(h)} - l_L)^2}{gL_S} - \frac{gL_S}{4} \tag{7}$$

$$d_H = \frac{d_1^2}{L_{B(h)} + 0.45d_1} \tag{8}$$

The algorithm for tuning the model at a tonehole position given a desired frequency for that tonehole is outlined below:

1. Calculate the value of the effective length $L_{S(h)}$ for this tonehole.
2. Calculate the value of g from the ratio of the desired frequency in relation to the ratio of the frequency of the lower adjacent tonehole as detailed in section 2.3.
3. Start with a reasonable value for d_H that is above a minimum threshold value. Compute $L_{B(h)}$

using equation (7).

4. Solve for z using equation (4).
5. Compute l_H using $l_H = L_{S(h)} - zL_{S(h)}$.
6. Compute d_H using equation (8).

The resulting values of d_H and l_H are appropriate parameters to tune the tube at the tonehole position.

The inclusion of minimum threshold values for d_1 and d_H is the result of observing detuning effects that occur when tonehole diameter values are too small. Small radii translate to small tonehole filter coefficients, which cause filters to have less effect on the signal passing through them. This decreases the effect that the tonehole has on the pitch of the model–small enough coefficient values have been observed to reflect the signal almost entirely. This behavior is aligns well with intuition: the smaller the tonehole radius, the less the output pitch will decrease when the tonehole is closed. If the tonehole is small enough, the pitch difference between opening and closing the tonehole could become unnoticeable, and the tonehole may as well not be part of the instrument. The exact minimum threshold at which unwanted detuning begins to occur is unknown, but values of 1.0 (in sample lengths) for both variables were deemed adequately effective in the Birl model.

An additional source of detuning was observed at the fundamental. For most fundamentals, tuning using the above algorithms almost always resulted in fundamentals that were audibly sharp. The exact cause of this is unknown, but Forster acknowledges this effect and notes that equation (2) is not an exact equation, but rather a "good estimation" for the relationship between d_1 and Δl_T, and furthermore "an exact equation for the end correction of open and closed tubes does not exist because the end correction depends slightly on wavelength" [2]. A practical solution has been found by tacking on an extra dummy tonehole at the end of the bore that is beyond the reach of the user. It seems that the tuning equations at tonehole positions are somewhat more accurate than the equations for tuning the main bore parameters. Adding on more than one dummy tonehole did not produce a perceivable difference from using only one dummy.

To fully unlock the potential of the physical model, tonehole radii and bore radius adjustment

features were added to the model's ChucK interface. Once a tuning system is chosen, users can further modify the model by tweaking the radius lengths along a range of suitable values. This provides the user with a higher level of control, and allows for the possible discovery and use of more unconventional instrument tunings. Unfortunately, in the prototype's current form adjustment can only take place during set up and not in real time during use. This is because the filter coefficients for that tonehole must be recalculated, and thus the tonehole coverage values become reset.

3.2.5. Additional Timbre Effects This section describes some additional signal processing effects that were added to the signal chain of the model. These effects are not part of the design of the underlying acoustic tube physical model and are by no means necessary to construct such a model. However, the model provides new opportunities for experimentation with effects *inside* of the model's signal chain, as DSP objects added inside of the model's computation loop will produce different effects than if they are added as post-processors outside of the model.

- Waveshaper [12]: A waveshaper was added at the bell location. The bell-reflected signal is first passed through this shaper before being input into the delay line moving towards the mouthpiece direction. The shaper has two separate mix and drive controls.
- Noise bandpass filter: A bandpass filter was applied to the noise component of the breath pressure inputs to the system. This filter provides subtle coloring effects to the timbre of the instrument. The filter is implemented as a state-variable filter following the code and specifications from [18]. The filter has three separate gain, cutoff frequency, and Q controls.
- Lowpass biquad filter: The Birl model on its own often produces an audible amount of noise in the upper range of the frequency spectrum. The noise is relatively jarring, and to tame this effect a lowpass biquad filter was added at the beginning of the signal chain. The output of the reed table lookup is passed through this biquad before being input into the delay lines that make up the model's body. Initially the STK Biquad class was used, but later on a custom biquad was implemented from the tutorial in [15].
- Sweepable peak/notch filters: Attempts were made to add a sweepable peak or notch filter to the model's signal chain. The filters were implemented as state-variable filters, again drawing

the implementation code from [18]. However, since the filters were being applied to the breath pressure signal inside of the model, noticeable detuning effects resulted that varied with cutoff frequency. The detuning effects of the filters were ultimately deemed too unpredictable for them to be useful in a performance setting, and the filters are not included in the current prototype.

3.2.6. Interface Documentation/API A brief listing of the model's ChucK-facing API is given below. For variables that take two arguments, the first argument is usually the tonehole or tube index. The indices are 0-indexed and 0 corresponds to the tube or tonehole closest to the mouthpiece.

- .breathPressure - (float, WRITE) - Set input breath pressure value.
- .length - ((int, int), WRITE) - Set tube segment length to value of second argument.
- .toneHole - ((int, float), WRITE) - Set tonehole coverage to value of second argument. 0.0 corresponds to fully closed, 1.0 corresponds to fully open.
- .toneHoleRadius - ((int, float), WRITE) - Set tonehole radius to value of second argument. Radii values must be within a pre-defined range (0.0001-0.004 sample lengths).
- .tweakToneHoleRadius - ((int, int), WRITE) - Adjust a tonehole's radius by a pre-defined tweak factor multiplied by the second argument.
- .tweakBoreRadius - (int, WRITE) - Adjust the main bore radius by a pre-defined tweak factor multiplied by the input argument.
- .setTuning - (int, WRITE) - Choose a tuning system according to these specifications:
  0. Equal-tempered.
  1. Just intonation.
  2. Meantone.
  3. Highland bagpipe (tuning ratios taken from [13]).
  4. Custom tuning. To use this tuning, the user must first call .setCustomTuning. Otherwise, this option is equivalent to the equal-tempered option.
- .setCustomTuning - ((float, float, float, float, float, float, float, float, float, float, float), WRITE) - Define a custom tuning by specifying a set of 11 frequencies that make up the tuning. The highest pitch should appear first. The lowest pitch should be one scale-step below the desired fundamental

(this is to accommodate the dummy tonehole hack described in section 3.2.4). Currently the system requires 11 frequencies even if the user is using less than 11 pitches in the model because the tuning buffer must be filled up to the maximum number of possible pitches in the model. This functionality can be improved in the future.

- .setFundamental - (float, WRITE) - Set the fundamental frequency against which the instrument will be tuned. The fundamental frequency is achieved during performance by covering all toneholes.
- .shaper - (float, WRITE) - Drive parameter of waveshaper function. Values are clipped between 0.0 and 1.0.
- .shaperMix - (float, WRITE) - Mix parameter of waveshaper. Values are clipped between 0.0 (dry) and 1.0 (wet).
- .noiseGain - (float, WRITE) - Gain multiplier for amount of noise added to input breath pressure signal before being input to the physical model. Values are clipped between 0.0 and 1.0.
- .noiseBPCut - (float, WRITE) - Cutoff frequency for bandpass filter applied to noise.
- .noiseBPQ - (float, WRITE) - Q factor for bandpass filter applied to noise. The Q factor corresponds roughly to the inverse of the filter's bandwidth. Lower values imply a wider bandwidth.

4. Evaluation

Evaluation of the Birl physical model involves determining whether the model can reproduce some wind instrument acoustic properties (section 4.1) and examining the practical usability of the model in a performance setting. As part of the evaluation phase the Birl was given to a trained saxophonist who was asked to explore the Birl and evaluate the acoustic effects produced by the model. Section 4.2 discusses an undesired global detuning observed in the model in its current form.

4.1. Acoustic Properties

We used four key acoustic properties unique to wind instruments as evaluation metrics for the Birl physical model. They are listed below, followed by a discussion of the quality of their replication in the model. Recordings of the effects can be found on this project's github repository.

- Half-holing: The tonehole models outlined in [4] and [17] are engineered specifically to allow for coverage values between the extremes of fully open or fully closed, and therefore the Birl model excels at reproducing this effect. Quality of response depends on the controller.
- Cross-fingering: This is a technique for decreasing the pitch at a tonehole i by opening i and closing all toneholes below i. Even though opening a tonehole shortens the effective length of the tube, the standing wave propogates beyond the tonehole opening and can therefore be affected by the length and characteristics of the part of the bore that extends past the open tonehole [19]. Since the Birl physical model accurately replicates the structure of a real instrument, cross-fingering does work intuitively. However, the exact amount of pitch decrease is difficult to determine.
- Overblowing: Overblowing occurs when a player blows strongly into the instrument, resulting in a frequency jump from the fundamental to an overtone of the fundamental. Overblowing in the Birl is not completely intuitive when compared to a real instrument. Simply blowing strongly into the mouthpiece is not enough to achieve overblowing–a particular fingering configuration must also be incorporated. During testing, the participating saxophonist was not able to produce overblowing on demand, but accidentally brought the model into an overblown state inadvertently by experimenting with the toneholes on multiple occasions. The model stayed in this state and continued to produce higher-frequency content until the point at which the player stopped blowing. In this state, the player was able to continue playing the instrument, and changes to tonehole configurations produced intuitive corresponding changes to the output tone.

  A hypothesis for this result is that the combination of high breath pressure coupled with some combination of filters (which are models of the toneholes) is what causes the model to enter a state where it resonates more strongly at overtones instead of at the fundamental. The exact filter configuration that causes this is unknown, and although overblowing is possible with the model,

it is still too unpredictable to be useful in a performance setting.

- Multiphonics: Multiphonics is the production of two or more tones that are part of separate harmonic series, and thus perceived by the audience as two distinct tones. The exact method of producing multiphonics in woodwind and brass instruments is somewhat unpredictable and varies greatly between different players and instruments. Although the participating saxophonist was able to produce multiphonics with the Birl physical model, the results were unreliable and there was no way to determine which tones were going to be produced. However, due to the unpredictable nature of multiphonics in physical wind instruments, this behavior actually aligns relatively well with the production of multiphonics in physical instruments.

4.2. Detuning

The single most obvious defect of the Birl physical model is a global pitch offset that is unsolved at the time of writing. While the tuning system described in section 3.2.4 correctly tunes the model relative to the specified fundamental, there is an observed noticeable discrepancy between desired pitch set by the user and the frequency of sound output by the model. Table 1 gives these differences for some specified frequencies.

The cause of this offset is unknown, but two hypotheses are discussed here.

Specified Pitch (Hz)	Output Pitch (Hz)	Difference (Hz)
100	116	16
200	223	23
300	319	19
400	408	8
500	511	11

Table 1: Discrepancies between some desired input frequencies and actual frequencies output by the model.

- The physical model necessitates filters in some key locations. Specifically, a lowpass filter is part of the signal chain at the bell reflection location, and the input breath pressure is also lowpass filtered to reduce unwanted high-frequency content. However, since the filters are acting inside of the model (as opposed to post-filtering the model's output signal), they have noticeable effects on

the pitch of the model. This is because the frequencies at which the tube resonates will depend partially on the input signal's frequency content as well.

Part of the reason sweepable pitch or notch filters were omitted from the signal chain (see section 3.2.5) were because the output tones of the model changed significantly when the cutoff frequencies were modulated. The pitch varied too widely for the effect to be useful. Even with static cutoffs, filters inside the model will have an effect on the pitch.

- Equation (3) is known to be an estimation, and an exact solution to calculating the end correction of an acoustic tube is unknown since the end correction and wavelength are interdependent. The repercussions of basing the tuning of the fundamental on this equation are an observed detuning of the fundamental tone (see section 3.2.4). Even though the immediate consequences of this problem were ameliorated with the addition of a dummy tonehole past the last tonehole, this equation is one of the first steps in the tuning algorithm we use, and any errors it introduces would be propagated throughout the rest of the system. The value of the bore diameter d_1 calculated in this step is used in all of the remaining tuning equations for the toneholes. It is possible that since a large part of the tuning system depends upon this equation, its accuracy is of more importance than previously thought, and new methods must be employed to refine its correctness.

Some ideas for working around this pitch offset are offered in section 5.

4.3. General User Feedback

Wind instrument players familiar with their instruments will quickly notice that one common wind instrument feature is omitted from the Birl: a register vent. Vents are often used to push the output tones of an instrument up by one octave. Furthermore, the BlowHole STK class implements a vent as well. The reason we were unable to implement a vent is because the BlowHole model is optimized into a version that uses a single delay line, and we were unable to correctly translate this model into the dual-delay line bidirectional version that we used in the Birl model, nor implement the "two-port scattering junction" for a vent described in [17].

A practical method to allow octave jumps is to run two concurrent Birl models, one with a fundamen-

tal one octave higher than the other, and allow the user to switch between the two. A crude prototype of this method can be found in the file `Birl2Octaves.ck` on the project github repository.

As mentioned in section 4.1, a saxophonist was invited to explore the Birl while it was connected to the physical model implemented in this project and asked to evaluate the experience. He noted that the multiphonics generally felt similar to the way that they are achieved in a saxophone, by using a "somewhat unpredictable embouchure/air stream" in conjunction with certain cross-fingerings. Overall, the acoustics felt "very authentic." However, he also mentioned that the instrument felt " a little too sensitive" and that it was hard to know when the model would start squeaking. Half-holing was also relatively unpredictable, and the biggest area of improvement was in giving the user more control over the sound output by the model.

5. Further Work

The Birl physical model implemented as part of this project represents a useful step in the direction of full-instrument wind instrument models with continuous tonehole control, but there is still much to be accomplished within the project. Furthermore, the Birl physical model is relatively simple, and can be easily extended. Some ideas for the expansion and development of this project are presented here.

- Solving the global pitch offset problem: The detuning problem discussed in section 4.2 is the most obvious current shortcoming of the model. Development topics relating to this issue include:
  - Further research: Further research must be conducted to determine the cause of the pitch offset. Hypotheses presented in section 4.2 suggest that it may have something to do with the approximating nature of equation (2), and the development of a more accurate equation for calculating the bore end correction may help solve this problem.
  - Automated lookup table: Some of the detuning is known to be caused by filters, and if the filters are to be included in the model, a certain amount of detuning is unavoidable. A practical solution to the detuning is to request a pitch that is offset by the correct amount from the actual desired pitch. For example, from table 1 we know that to produce 408 Hz, the user

must request 400 Hz from the model. We can generalize this hack by building a lookup table that stores the mappings from desired pitch to output pitch. An automated process could be implemented that uses sinusoidal decomposition of the model's output to determine the actual frequency for every requested frequency.

  - Develop tweaking functionality: One approach to the pitch offset problem is to ignore it and place the responsibility of getting the model in-tune with other instruments entirely on the user. This correlates with reducing the requirements of the model from absolute tuning to relative tuning. However, the user needs a good interface to adjust the instrument in order to accomplish this, and the bore and tonehole radii adjustment system currently implemented is somewhat primitive. It does not allow the user to tweak the radius parameters in real time. Instead, the user must stop and start playing and re-cover the correct toneholes in order to hear the effects of any adjustment. This is because after a parameter change, the tonehole filter coefficients must be recalculated. A solution to this is to store the coverage values of each tonehole in a buffer, and when tweaking occurs, re-initialize the values of the toneholes so that they are consistent with the contents of the buffer.

- Absence of vent: Further research could be performed to determine how to implement the "two-port scattering junction" in a dual-delay line model as outlined in [17].
- Interface design/distribution: The development of an intuitive user interface and porting the model to a different digital audio framework could make it more attractive for new users, and encourage existing electronic music communities to experiment with it. Good candidates for alternate frameworks include Max/MSP, Pure Data, Faust, and the VST plugin format.
- Translation to C: Ultimately, the goal of the Birl project is to run the Birl as an embedded system that synthesizes its own output. The microcontrollers currently embedded in the Birl can only execute C code, so a necessary step before this goal can be achieved is to translate the prototype from the C++ ChuGin framework into pure C code. Efforts have already been taken to remove the prototype's dependence on the C++ STK library, but most of the code still includes C++ specific syntax.

- Profiling/Optimization: Performance benchmarks of the prototype can be calculated in order to determine how well it can perform on the Birl's microcontrollers, as well as other processors. Optimization similar to that undertaken in the STK BlowHole class can also be performed, but further research may be required in order to determine how this must be implemented.
- Complex models: The current prototype models a relatively simple clarinet, but it can be used as a foundation for implementing more complex instruments and physical systems, such as saxophones, flutes, oboes, and other brass and woodwinds.
- Control: Further research to give more control to the user in terms of generating acoustic effects such as multiphonics. For example, connecting half-holing of a tonehole to generating a multiphonic related to that tonehole.

6. Conclusion

The Birl physical model represents an early step towards construction of the bridge between the digital and physical realms with respect to wind instruments, and provides a foundation off of which more complex models can be developed. It brings current technology closer to what is known as *parity*, or interchangeability, between virtual and acoustic wind instruments [8]. On top of that, it offers a new creative space to be explored by artists. By allowing artists direct control over the physical parameters and specifications of their instrument, new sounds and timbres can be created by fusing the timbres connected to traditional instruments with tweaks and customizations that are beyond the capabilities of physical instruments.

In a 1996 paper [8], Julius O. Smith (whose prolific work in the field of physical modeling and digital signal processing has been cited numerous times in this paper) noted that his students often asked him what the importance of modeling traditional instruments was when a computer is capable of producing a far wider range of sounds. Why do we spend time attempting to replicate the physical characteristics of traditional instruments, when such a wide and exciting range of sounds can be produced with digital synthesizers that depart completely from the traditional realm of music-making? Grimy dubstep bass wobbles, luscious trance pads, and massive 808 trap kicks that

form the basis of the modern popular electronic music scene are all synthesized from simple saw, square, and sine wave oscillators, sounds which are not found in any physical instrument.

Smith's answer is twofold. Firstly, "artificially computed sounds tend to sound artificial" and "we simply don't know very many ways to generate deeply communicative sounds from scratch." Furthermore, "traditional musical instruments are important because they are recognizable." In other words, our experience with music depends in a large part on our previous experiences and memories associated with the pitches and timbres we perceive. In this way, traditional musical instruments will always have a place in our collective listening history, and the ability to reproduce the timbres associated with those instruments is a connection between that history and the electronic music world. Physical modeling synthesis, therefore, represents a bridge between the physical and digital worlds for music. It allows unprecedented levels of expressivity to be possible with electronic instruments, and provides the user with levels of control and customizability that are free from limitations imposed by a physical instrument.

References

[1] J. Chong, "A rule-based fingering system for the birl electronic wind instrument," in *Princeton University Independent Work*, 2014.

[2] C. Forster, *Musical Mathematics*. Chronicle Books, 2010.

[3] M. v. W. Gary Scavone, "The wave digital tonehole model," in *International Computer Music Conference, Berlin, Germany*, 2000.

[4] P. C. Gary Scavone, "Real-time computer modeling of woodwind instruments," in *International Symposium on Musical Acoustics, Leavenworth, WA*, 1998. Available: https://ccrma.stanford.edu/~jos/tonehole/tonehole.html

[5] P. C. Ge Wang. (2002) Chuck : Strongly-timed, concurrent, and on-the-fly music programming language. Available: http://chuck.cs.princeton.edu/

[6] S. S. Ge Wang, "Chugens, chubgraphs, chugins: 3 tiers for extending chuck," 2012. Available: https://ccrma.stanford.edu/~spencer/publications/CCC2012.pdf

[7] J. O. S. III, "Digital waveguide architectures for virtual musical instruments," 1993. Available: https://ccrma.stanford.edu/~jos/asahb04/asahb04.html

[8] J. O. S. III, "Physical modeling synthesis update," in *The Computer Music Journal*, 1996. Available: https://ccrma.stanford.edu/~jos/pmupd/pmupd.pdf

[9] J. O. S. III, *Physical Audio Signal Processing for Virtual Musical Instruments and Audio Effects*. W3K Publishing, 2010. Available: https://ccrma.stanford.edu/~jos/pasp/pasp.html

[10] J. S. J. Wolfe, N.H. Fletcher, "The interactions between wind instruments and their players," in *Acta Acustica United with Acustica*, 2015. Available: http://newt.phys.unsw.edu.au/jw/reprints/Wind-Instrument-Overview.pdf

[11] D. R. Jeffrey Snyder, "The birl: An electronic wind instrument based on an artificial neural network parameter mapping structure," in *International Conference on New Interfaces for Musical Expression, Goldsmiths, University of London, UK*, 2014. Available: http://www.nime.org/proceedings/2014/nime2014_540.pdf

[12] R. Jones. Waveshaper.

[13] E. Macpherson, "The pitch and scale of the great highland bagpipe," in *New Zealand Pipeband*, 1998. Available: http://publish.uwo.ca/~emacphe3/pipes/acoustics/pipescale.html

[14] G. S. Perry Cook. (1995) Stk. Available: https://ccrma.stanford.edu/software/stk/index.html

[15] N. Redmon. (2012) Biquad c++ source code. Available: http://www.earlevel.com/main/2012/11/26/biquad-c-source-code

[16] S. Salazar. (2012) Chugins. Available: https://github.com/spencersalazar/chugins

[17] G. Scavone and J. O. S. III, "Digital waveguide modeling of woodwind toneholes," in *International Computer Music Conference, Thessaloniki, Greece*, 1997.

[18] A. Simper, "Solving the continuous svf equations using trapezoidal integration and equivalent currents," 2013. Available: http://www.cytomic.com/files/dsp/SvfLinearTrapOptimised2.pdf

[19] J. Smith and J. Wolfe, "Tone holes and cross fingering in wood wind instruments," in *International Congress on Acoustics, Rome*, 2001. Available: http://newt.phys.unsw.edu.au/jw/ICASmith&Wolfe.pdf

[20] J. Snyder, "Exploration of an adaptable just intonation system," in *Columbia University Graduate School of Arts and Sciences Doctoral Dissertation*, 2010. Available: http://www.scattershot.org/full-dissertation.pdf