Qingyang Xi
DST Fall 2015
Final Project

Matlab Classes for Physical Modeling Using Digital Waveguide

Physical Modeling Synthesis has been attracting more and more attentions in the past decade, albeit the fact that the developmental efforts on physical modeling can be traced back to Hiller and Ruiz in 1971. Early approaches to physical modeling are variations of using a difference approximation to solve for the wave equations numerically. By supplying different initial values to the Partial Differential Equation (PDE) that describes the state of a vibrating body, different sounds can be generated. Numerical methods of this type were computationally exhausting in the 20th Century and was certainly an interest only to academics. Many other methods that attempts to improve computational efficiency has been developed over the years, striving towards real time algorithms. Among these attempts is the Digital Waveguide (DWG), a method invented by Julius Smith at CCRMA in the 1980s.

In short, Digital Waveguide uses clever combinations of digital delay lines to simulate a vibrating string, and turns out to be both elegant and computationally efficient. This project focuses on an conceptual implementation of the DWG method using Matlab classes. Two essential objects in the DWG method are the Bidirectional Delay Line (BDL), which is analogs to a vibrating string, and the Scatter Junction (SJ), which serves as a way of to connect BDLs or to terminate them. Two classes called qx244_bdl and qx244_sj are created to realize the implementation, which reflects the theory of DWG. The rest of this paper will discuss conceptually the theory of DWG and justify and explain the structure and method of the two classes implemented, along with some demonstrations.

A disclaimer should be included at this point that comments on the speed of the implementation. Although the DWG boast computational efficiency, my implementation was designed to clearly illustrate the *concept* of the algorithm, as opposed to being computationally optimal. Therefore, the speed at which this implementation synthesizes sounds is not typical of a DWG; they are quite slow, but nonetheless they still work as a sandbox for initial investigation.

The theory of DWG rests on the D'Alembert solution, or traveling wave solution, to the wave equation, which states the behavior of any ideal 1D object that obeys the wave equation can be decomposed into the sum of a right-going and a left-going traveling waves. Traveling waves are waves that propagates through the medium at a constant speed without changing shapes; to visualize a traveling wave, imagine a stone thrown into a found, the circular ripples radiating out from the point of contact are in fact traveling waves in all directions, and the behavior of the water surface can be seen as the sum of all these traveling waves. Since ideal traveling waves doesn't change shape and travel at constant speed, a traveling wave can be modeled by a delay line. Since any solution to the 1D wave equation can be expressed by summing two traveling waves in opposite directions, a pair of delay line can simulate the behavior of a string exceedingly well and efficiently. Such a pair of delay line of the same length and propagating in the different direction is called a bidirectional delay line, or BDL.

The class qx244_bdl uses two row vectors dLineR and dLineL to represent the two delay lines simulating traveling waves. A BDL has two parameters, its wave impedance, a parameter that will determine the behavior of the scatter junction when it is connected to one, and the length of the delay line, representing the length of the actual string the BDL is simulating. The rest of its properties are there to facilitate its connection with a Scatter Junction, which will be discusses later. The bdlDemoScript file demonstrates all but one of the methods associated with qx244_bdl. The one method that is not included in this script is the .connect() method, which is designed to work with a Scattering Junction object. The bdlDemoScript should prove efficient in demonstrating the concept and power of the qx244_bdl class.

Associated with the last section of the bdlDemoScript is the .tap() method. The concept behind this method deserves a short explanation. While the BDL simulates the state of the vibrating string, it cannot be readily converted to a sonic ouput; after all, the entire string is presented as a vector at all times in this simulation scheme. The tap object functions as a pickup mic on the delay line at a specific shift register, and when the output of the .tap() method is recorded over time, the resulting vector is the vibration of a point on the string over time, and can be used as a sonic vector. The schematic of the .tap() method is illustrated in figure 1, on the next page.

At this point, the BDL structure cannot generate interesting sounds yet, and this is because the current model can only model ideal and unterminated strings. The effect is like throwing a stone into a endless pond, and the ripple traveling outwards never reflect back. However, It is these behavior when the propagating traveling wave is faced with a termination that interesting interaction happens. The rest of the paper will attempt to simulate a common guitar string, which is rigidly terminated on both side.

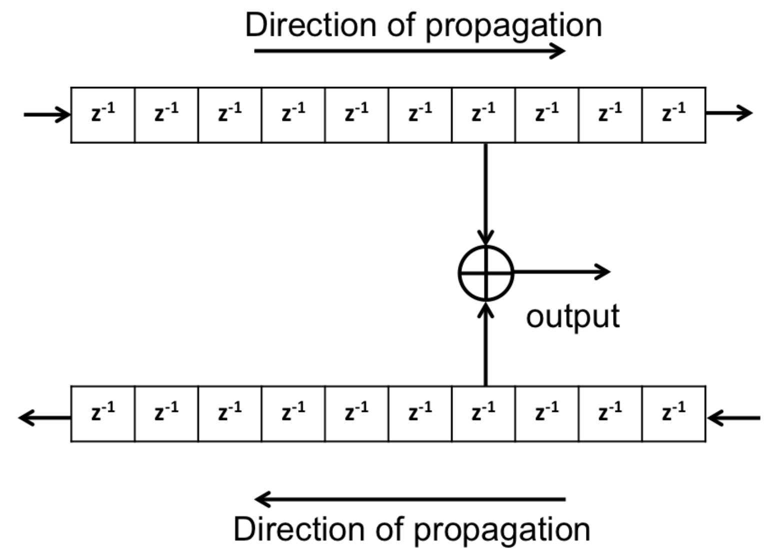

Figure 2: Block diagram illustrating the .tap() method for qx244_bdl class

Before explaining the implementation of the class for Scattering Junction, the concept of wave impedance needs to be discussed. It should be first noted that up until now, all of the simulations assumed the wave that is being passed around in the BDL are numbers that represents the deviation in *displacement*. However, traveling waves in a BDL doesn't have to be displacement waves: if we take the first spatial derivative of the signal, it would become a slope wave, or if we that the first time derivative, it would be the velocity wave. One could also place quantities that represent deviation from the standard pressure in a BDL, effectively sending pressure waves around. This is the approach that allows DWG to model wind instruments. Evidently, there is a choice of different wave variables available for simulation when the system is setup. Through some manipulation of the 1D wave equation and its traveling wave decomposition, the traveling force wave can be related to the traveling velocity wave of a vibrating object. The relationship, also known as the Ohm's law of traveling waves, can be expressed as the following equations: $f^{+}=R * v^{+}$ and $f^{-}= -R * v^{-}$, where R is called the wave impedance. In the case of a vibrating string, wave impedance is a function of string tension and string density.

Traveling waves will only behave nicely on a string if the medium is consistent, in other words, with a constant wave impedance through out. As soon as the traveling wave is faced with a change in impedance, only part of the wave will be propagate into the new medium while parts of the wave reflect back into the original medium. This phenomenon is known as wave scattering. Scattering of the traveling wave at impedance change is a well studied phenomenon, and one only related to the impedance of the elements connected. The formula that is used in the implementation of qx244_sj is taken from Julius Smith's tutorial on the subject, thus a detailed derivation is not included here. A rigid termination on the string is a extreme example of scattering, where all of the wave is reflected back. Scatter Junction can be loaded with a impedance, just like a BDL, and the implementation supports this functionality.

Most of its methods are intended to be used by a qx244_bdl instance to facilitate connection, and is therefore not demonstrated. To connect BDLs with Scattering Junctions, use the .connect() method that's associated with the BDL class. The bdl.connect() method takes two arguments: the scattering junction it wants to be connected to, and a string ('r' or 'l') to indicate which side of the BDL should be connected. Once a connection is made by calling the function, the sj class property ports (a list of BDLs connected to the SJ) and Rsum (sum of Impedances) are updated in the SJ instance, and the output of the SJ and BDL are automatically routed to the right place. The current implementation doesn't check if one side of a BDL is connected to multiple SJs, a condition that is physically impossible, and would cause errors in simulation.

The a basic wave scattering at impedance change is demonstrated in the sjDemoScript file. Two BDLs and one SJ are created and connected in the script, then the simulation steps through time to simulate the scattering phenomenon. Although the script only connected two BDLs to the SJ, the implementation doesn't have a restriction on how many objects are connected to one Scattering Junction. One can extend the system constructed in sjDemoScript by creating another BDL and connecting it to mySJ, and the scattering would still function properly. This is an advantage to this implementation of DWG that is not enjoyed universally.

Rigid termination can be simulated by using SJs with high junction impedances. In pluckedStringDemo, such a system of rigidly terminated string is constructed and the simulation generate a 1 second long sound example from the setup, initiating the BDL with noise. The script has 3 parameters that the user can modify, and they would have an impact on the resulting sound produced. Parameter Rj is the impedance of the the two Junctions used to terminate the string segment. An ideal rigid termination would have infinite impedance, and would reflect all of the waves energy back, causing no decay in the sound output. Varying the Rj variable would give a sense of how the termination condition affects the system. The pitch of the simulated sound is directly related to the stringLen variable, just like in the real world. This variable controls the length of the BDL and doubling this number would lower the generated pitch by an octave. The location of the pickup can also be specified by the pickUpPoint variable. The initial settings attempts to imitate the guitar, where the pickUpPoint at 20 is about

1/3 of the length of the string, which is 70. Changing pickUpPoint to a different spot while keeping everything else the same will generate sound that vary slightly in timbre, just like in the real world. The script runs surprisingly slowly, probably because the delay lines are constructed out of vectors and extra copies of these vectors are made carelessly along the way of my implementation. Nevertheless, the simulation does finish within about 20 seconds.

This concludes the explanation of the classes that I've created to illustrate concepts in Digital Waveguide. Albeit the disappointing speed at which the implementation runs, these classes that abstract the BDL and the SJ can be useful tools for educational purposes. The simplicity of the API and the power of the system is thoroughly demonstrated in this paper and the accompanying scripts, along with a simulation that generates a sound of a plucked string.

Sources Consulted

Bilbao, S. (2009). Numerical sound synthesis: Finite difference schemes and simulation in musical acoustics.

Smith, J.O. Physical Audio Signal Processing. (2010).
Available at http://ccrma.stanford.edu/~jos/pasp.html, 2015.