Assignment 3: Real-Time Implementation of 2D Digital Waveguide

ECS7012P Music and Audio Programming

Andrea Martelloni

May 15, 2020

Abstract

Sample abstract.

1 Introduction

A specific class of digital musical instruments, or DMI, aims to decouple the human playing interaction and the physical response of the acoustic body. The instrument is thereby seen as a mute controller interface to a digital sound-producing device. This design paradigm allows the control of virtual instruments based on physical models. It might be seen as bizarre to choose to cut off the physics of an instrument, only to then try and reproduce them as accurately as the technology allows. However, the designer can then alter or augment the physics of the instrument arbitrarily; for example, they could map a drum or a set of percussions onto a guitar's body.

The work presented in this report aims to cover the first steps in the design of such an instrument. We are going to try and run a virtual vibrating membrane on Bela, and control it with the signal coming from one axis of an accelerometer.

The major challenge of this task is to ensure a correct implementation of the membrane's equations, a requirement that can prove very tricky when programming on an embedded target as a black box. Therefore, most of our analysis will be performed in simulation on standard C++ code; on-target evaluation will be the last step, a rather ambitious moment of truth at the end of the development efforts.

2 Background

2.1 Prior Work

The numerical reproduction of percussion instruments has been extensively studied, all the way to advanced techniques modelling the non-linearities of such systems. Good surveys of all available techniques come from the works by Rossing *et al.* [Rossing, 2004], Bilbao [Bilbao, 2009] and Mehes *et al* [Mehes, 2017].

Implementations of purely linear model are both less computationally intensive and easier to understand and implement. Two methods seem to have been quite successful in reproducing membranes in real time and are featured in working implementations of musical instruments: the Digital Waveguide model by Fontana and Rocchesso [Fontana, 1998] and the Transfer Function methods analysed in Trautmann *et al.* [Trautmann, 2001].

The two-dimensional Digital Waveguide model is a well-established model that extends the principle of the Digital Waveguide, widely used to model tubes such as wind instruments, to a vibrating surface. The article referenced above provides a clear breakdown of all the continuous-time and discrete-time physics involved, and it provides solutions to problems such as mesh stability, mesh excitation, energy attenuation and modelling of air loading for a more accurate real-world drum reproduction. Thererfore, we chose to base the implementation of our project upon the Digital Waveguide method.

A reference implementation of a Digital Waveguide mesh is already available in an early version (0.62)[1] of the Sound Design Toolkit [Baldan, 2017]. The topology of the mesh available in the toolkit is rectilinear; however the authors describe how using a triangular mesh optimises the dispersion error for most geometries, and especially for the circular membranes of drums [Fontana, 1998]. We will implement a triangular mesh from first principles following this suggestion.

2.2 Theory

The following summary of the theory behind digital waveguides in a triangular mesh structure is based on [Fontana, 1998]; some ideas and concepts are taken from the implementation of triangular membranes for room acoustics in [Murphy, 2000]. A more complete review of the whole process applied to virtual drums can also be found in [Laird, 2001].

Waveguide basics. The principle of the one-dimensional waveguide comes from the well-known D'Alembert solution of the wave wave equation for the transverse velocity $v(t,x)$, indicating two velocity waves travelling in opposite directions:

$$v(t,x) = v_+(x - ct) + v_-(x + ct) \tag{1}$$

We can assume that waves propagate following the equation above in ideal strings. When a number N of identical strings is joined together, we can write an equation for the transmission and scattering of the waves among those strings at their junction point i:

$$v_{i-}(t) = \frac{2}{N} \sum_{k=1}^{N} v_{k+}(t) - v_{i+}(t) \tag{2}$$

v_{i-} being the outgoing wave from the junction, as a function of the incoming waves v_{k+} from the strings joined together.

Both equations can be discretised by sampling the space into Δs spatial samples and time as $t = nT$ periods:

$$v(nT,x) = v_+(x - cnT) + v_-(x + cnT) \tag{3}$$

$$v_{i-}(nT) = \frac{2}{N} \sum_{k=1}^{N} v_{k+}(nT) - v_{i+}(nT) \tag{4}$$

We are going to assume that the mesh is perfectly homogeneous (no difference in impedance in the interfaces) and the boundaries are perfectly rigid, that is, the wave is fully reflected back ($r = 1$) and any point beyond the boundary will have ($v = 0$).

[1] `http://www.soundobject.org/SDT/downloads/SDT_src-062.zip`

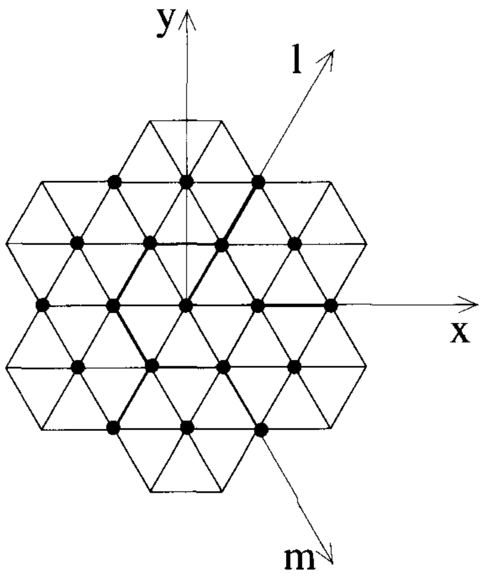


Figure 1: Triangular mesh coordinates [Fontana, 1998].

Triangular mesh. Joining six waveguides we have a structure that can effectively sample a two-dimensional space along directions x, $l = \frac{1}{2}x + \frac{\sqrt{3}}{2}y$ and $m = \frac{1}{2}x - \frac{\sqrt{3}}{2}y$. Figure 1 depicts this structure and the related coordinates.

We can then rewrite equation 2.2 in two parts and restrict the number of waveguides to a maximum of 6, assuming impedance is either homogeneous or infinite (no transmission). We can then divide the computation into a *scattering equation* and a *junction output* (after [Murphy, 2000]):

$$v_i(nT) = \frac{2}{N}\sum_{k=1}^{N} v_{k+}(nT) \tag{5}$$

$$v_{k-}(nT) = v_i(nT) - v_{k+}(nT) \quad \text{for}\, k = 1 \dots N \tag{6}$$

Junctions having fewer than six waveguides will just ignore the directions that aren't connected to another junction point. A good convention to enumerate the coordinates of the junction points comes from [Murphy, 2000]. We will count clockwise from the first point at top-right: North-East, East, South-East, South-West, West and North-West.

Constraints. The digital waveguide method imposes a relationship between spatial and temporal sampling. When designing the mesh or even running an offline simulation, one can leave the speed of the medium c unspecified. However, in a real-time system, the temporal sampling period T is defined by the system, and so must be the medium speed. The stability of the mesh is enforced by the Courant condition, which in this case takes the following form [Fontana, 1998]:

$$\Delta x = \Delta l = \Delta m = \sqrt{2}cT \tag{7}$$

Absorption, air loading, excitation. The sources referenced provide a good breakdown of all the more advanced problems around lossy junctions, air loading in a cylindrical drum as a spring-mass model, excitation models. We will limit ourselves to the simplest case of a lossless stable membrane, excited directly with an

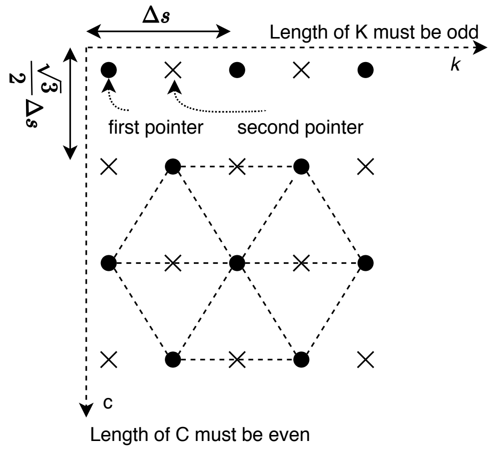


Figure 2: Mapping between two interleaved meshes and 2D array.

arbitrary audio signal $y(nT)$, using the excitation model outlined in [Murphy, 2000]. The source signal is injected as incoming waves into a designated junction:

$$v_{k+}(nT) = \frac{y(nT)}{2} \tag{8}$$

3 Design

The equations laid out in the previous section allow, in principle, the numerical implementation of any kind of topology. However, in addition to the memory constraints and computational constraints imposed by the Bela target, the triangular mesh poses non-obvious problems down to the memory layout and the change in coordinates between cartesian and (x, l, m).

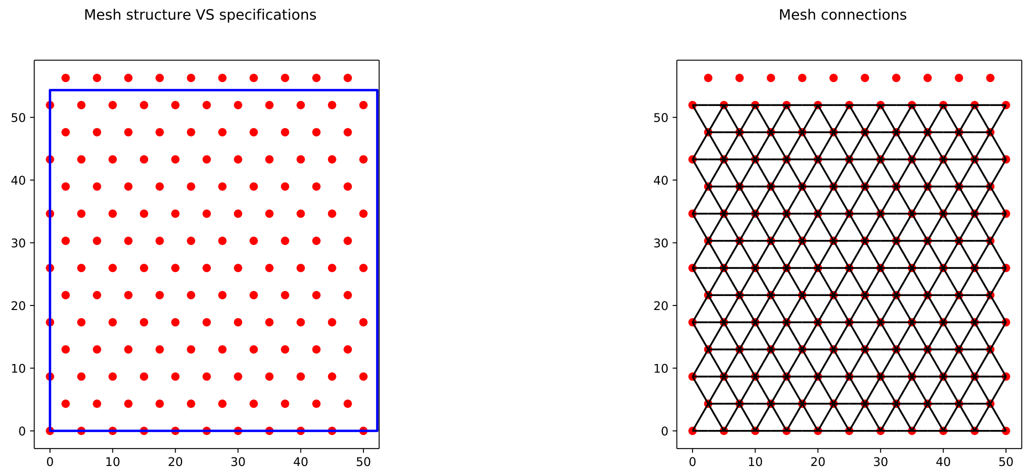
The first stage of the implementation, in practice, requires access to tools such as off-line processing, debugging, plotting and the generation of mathematically complex audio test signals. We found especially important to be able to visualise the mesh at every iteration in a simulation. For this reason, most of the code has been written within an environment comprised of some C++ core classes, a C++ to Python wrapper based on Boost.Python[2] and a set of validation notebooks written in Jupyter.

3.1 Mesh Topology

Rectilinear grids have an obvious mapping onto two-dimensional arrays, natively supported in most programming language. More complicated topologies require a function that maps a C++ array structure to the mesh's geometry coordinates, which we will call c, k. The idea of using a flattened array structure, albeit valid and quite well-established, came to mind a bit too late in the development.

We present a method that maps a regular C-style 2D array structure to a rectangular mesh topology, seen in figure 2. If we impose the constraints that the column dimension of a matrix must be even, and the rows must be odd, we can interleave two meshes of identical size in the even and odd indexes of this matrix, respectively. Even rows will have an odd number of mesh points, whereas odd rows will have an even number of them. What makes this structure work is that the third element of the second mesh in even rows is wrapped around to the next line, therefore the indexing stays within the required memory boundary.

[2]This has been the theme of a past AIL workshop at QMUL, the slides for which can be found here: `https://docs.google.com/presentation/d/1FK514Me0esJG-2egqraF3FT9DIQkZYGH_I2nMmFxUXI`



(a) Dimensions of the grid VS required geometry.

(b) Computed mesh connections.

Figure 3: Generated mesh for a rectangle of width 52.21 mm, height 54.35 mm, spatial resolution 5 mm.

The following set of equations can be used to convert back and forth from c, k indexing and Cartesian coordinates:

$$c = \frac{y}{\frac{\sqrt{3}}{2}\Delta s} \tag{9}$$

$$y = c\,\frac{\sqrt{3}}{2}\Delta s \tag{10}$$

$$k = \frac{x - \frac{\Delta s}{2}(c \bmod 2)}{\Delta s} \tag{11}$$

$$x = k\,\Delta s + \frac{\Delta s}{2}(c \bmod 2) \tag{12}$$

Using this coordinate system, we can then assess whether each point in the mesh lies inside or outside any geometry defined in Cartesian coordinates. We can then choose to create a waveguide between two points if both of them lie within the given geometry. An example made whilst evaluating our mesh in the Jupyter notebook is shown in figure 3.

3.2 Processing

The last step needed before implementing the processing routine is to calculate how many values we need to store for each mesh point and/or for each waveguide. Following the reference implementation in the Sound Design Toolkit [Baldan, 2017], we are going to use two values for each waveguide, respectively an outgoing and an incoming velocity value. The outgoing value will be reused as an incoming value in the following time iteration (delay step). Therefore we need a total of 13 meshes for each mesh point, one for each waveguide plus a record of the junction values that is going to be used for plotting.

A fourteenth mesh is going to be created to store a bitmask containing the connections between a given point and its neighbouring point. We're going to observe Murphy's convention of clockwise numbering from NE to NW.

Finally, we need to choose a source and a pickup point. At startup, the source is allocated at the centre of the membrane, whereas the pickup lies close to the edge of the default rectangular structure, as no mode is supposed to have a node at the corners of a shoebox shape.

All these actions are implemented in the constructor for the object `Triangular2DMesh`: it also provides a method to externally allocate the memory required, which is then passed as an argument to the constructor.

Optionally, the API allows to re-draw the mesh connections based upon a user-defined lambda expression evaluating the x, y coordinates of each mesh point. Source an pickup points can also be moved before or during processing.

`ProcessSample()` implements equations 5, 6 and 8 for every point c, k in the mesh and swaps the two mesh pointers for each iteration. The outgoing values are therefore consumed by the neighbouring points of each junction.

3.3 Evaluation Framework

4 Bela Implementation

5 Evaluation

5.1 Unit Tests

5.2 Simulation

5.3 On-Target Evaluation

6 Conclusion

References

[Rossing, 2004] Rossing, T.D., Yoo, J. and Morrison, A., 2004. Acoustics of percussion instruments: An update. Acoustical science and technology, 25(6), pp.406-412.

[Bilbao, 2009] Bilbao, S., 2009, September. A modular percussion synthesis environment. In Proc. of the 12th Int. Conference on Digital Audio Effects (DAFx-09), Como, Italy.

[Mehes, 2017] Mehes, S., van Walstijn, M. and Stapleton, P., 2017. Virtual-acoustic instrument design: exploring the parameter space of a string-plate model. In NIME (pp. 399-403).

[Fontana, 1998] Fontana, F. and Rocchesso, D., 1998. *Physical modeling of membranes for percussion instruments.* Acta Acustica united with Acustica, 84(3), pp.529-542.

[Murphy, 2000] Murphy, Damian Thomas, and D. M. Howard. *Digital waveguide mesh topologies in room acoustics modelling.* Diss. University of York, 2000.

[Baldan, 2017] Baldan, S., Delle Monache, S. and Rocchesso, D., 2017. *The Sound Design Toolkit.* SoftwareX, 6, pp.255-260.

[Trautmann, 2001] Trautmann, L., Petrausch, S. and Rabenstein, R., 2001, May. *Physical modeling of drums by transfer function methods.* In 2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 01CH37221) (Vol. 5, pp. 3385-3388). IEEE.

[Laird, 2001] Laird, J.A., 2001. *The physical modelling of drums using digital waveguides.* (Doctoral dissertation, University of Bristol).