VPS VIBRATIONS IN PHYSICAL SYSTEMS

DOI: 10.21008/j.0860-6897.2022.2.06
Vibrations in Physical Systems, 2022, 33(2), 2022206

Simulation of Infeasible Instruments in a Sound Synthesizer – Implementation and Control

Marek PLUTA

Department of Mechanics and Vibroacoustics, AGH University of Science and Technology, al. Mickiewicza 30, 30-059 Krakow, Poland

Corresponding author: Marek PLUTA, email: pluta@agh.edu.pl

Abstract Sound synthesis using mathematical modelling of musical instruments is a method particularly well suited for live performance using a physical controller. Depending on model complexity, it may be able to reproduce various subtle phenomena related to excitation and real time control of an instrument, providing an intuitive tool for a musician. A variant of physical modelling synthesis, referred to as the simulation of infeasible instruments, uses a model of an object that does not have a physical counterpart. Such model has some properties of a real object, which makes it still intuitive for a musician. However, other features, such as geometry, or material properties, are intentionally altered in such manner, that it could not function in reality. These infeasible features introduce new properties to the sound it produces. The study presents a few such models with a discussion regarding their implementation and control issues in a real-time sound synthesizer.

Keywords: sound synthesis, sound synthesizer, physical modelling, finite difference method.

1. Introduction

Contemporary sound synthesizers, able to produce musical sounds, are often considered a kind of musical instruments. Where they differ from traditional instruments is a precise, more fundamental control over a timbre of produced sound [1]. Various synthesis methods achieve this goal in different ways, allowing to target specific use-cases.

One of main use-cases is a live performance, where a synthesizer is producing sound in real time, and a musician uses a dedicated device to control its operation. The controller may simply be a piano-like keyboard. However, if a musician wants to achieve some specific kind of control over produced sound, usually for expressive purposes, other devices may be utilised. Some are based on existing instruments, other can mix features of various instruments with different, less conventional ideas like EEG, EMG, or computer vision.

While designing a synthesizer for the purpose of artistic sound experiments, it might be a good idea to provide it with intuitive control – meaningful for a musician familiar with various traditional instruments. It would allow to use and combine prior musical knowledge and performance practice to achieve new effects without a steep learning curve. A method of synthesis particularly well suited for such purpose is to design and operate a mathematical model of a musical instrument, or its part, and solve it numerically to produce sound. It is often carried out using the finite difference method (FDM) [2, 3], digital waveguides [4-6], modal synthesis [7-10], or – less frequently – the finite element method (FEM) [11]. In this approach parameters to control are the quantities describing a model or its operation [12]. Sound signal parameters are controlled indirectly, unlike in case of other synthesis methods. This, in turn, allows to easily map these quantities to controller gestures, if a controller is based on a similar instrument. The operator of a synthesizer can pluck, blow, strike, push, or touch, instead of adjusting amplitude envelopes and filter resonance, or fine tuning some more intricate characteristics.

Sound synthesis using mathematical modelling of musical instruments is a way to achieve realistic performances, with a natural-sounding expression. However, it can be extended to the area of sound experiments, while still maintaining advantage of intuitive musical control. If a model is altered up to the point where it could not function in reality, but still has some characteristics connecting it to the instrument it was based upon, it could be referred to as an infeasible instrument [1]. It is most natural to keep the excitation of a model similar to a real instrument. However, its geometry, dimensionality, material

properties, or movement patterns can be changed beyond physical plausibility, in order to introduce new characteristics to the sound it produces.

Initial ideas of this kind have been mentioned in works of Roads [13], Djoharian [14], or Leonard and Cadoz [15]. A motivation and formulation of assumptions, as well as several models of infeasible instruments have been discussed in Ref. [1]. The same work proposed a general technique of implementing some of more computationally-intensive models, such as multi-dimensional objects, using parallel processing. However, feasibility of operating and controlling such models remained to be proven. This paper presents an actual attempt to design and implement a synthesizer working in real time, based on the finite difference method, to simulate multi-dimensional membranes, and other objects based on a similar principle.

2. Models of instruments

2.1. Finite difference method

Among various methods of numerical simulation, the finite difference method (FDM) has advantages that make it a method of choice when it comes to sound synthesis. It approximates partial differential equations with finite differences, and restricts spatial domain of simulation to a grid with a finite number of points representing a displacement or other physical aspect [1]. Numerical solution is calculated recursively in discrete time steps. Every step produces a single signal sample, and any point of a grid can be considered its source, because the whole grid is accessible at any point of time. Similarly, it is possible to affect any point of a model at any point of time. These properties allow to design interactive models of instruments operating in real time.

In general, spatial and temporal sampling intervals of FD models are related through stability condition such as Courant-Friedrichs-Lewy (CFL) [16]. Thus for a given sampling frequency a synthesizer is required to work with, a model has to have at least a given number of spatial grid points. It increases computational cost of model operation in higher sampling frequencies.

Depending on the object modelled, various boundary conditions may be applied for objects that are e.g. clamped (fixed), open, or supported. Boundary conditions can be applied to model geometry of an object and some interconnections. Objects simulated with FDM may be excited to vibrate using various techniques, such as coupling with other vibrating or moving objects (e.g. mallets or bows), but simple excitation model can be implemented by applying particular initial conditions for grid points displacement or velocity, usually with some realistic spatial distribution.

2.2. Hyper-membrane

A membrane may be considered a two-dimensional extension of a string. Even though they differ only in dimensionality, they produce sounds with very distinct properties. Therefore, one might wonder, how would it sound if it was possible to add more dimensions to a membrane, creating an object that might be referred to as a hyper-membrane.

General properties of such hypothetical object were discussed in Ref. [1], where it was stated that N-dimensional membrane:

- has N+1 spatial dimensions,
- along one of its dimensions is small, while along the remaining N is large in relation to the wavelength,
- is clamped along all of its boundaries,
- is stretched in N 'large' dimensions, which provides elasticity,
- can be excited and vibrates along (N+1)-st 'small' dimension.

Even a three-dimensional membrane is physically infeasible. It would have to vibrate along fourth dimension, and its clamping along boundaries would make it inaccessible for excitation or for sound radiation outside. However, more interesting objects begin with four-dimensional membranes.

A generalised, simple N-dimensional membrane can be described using the wave equation supplemented with a simple loss model, such as [1]:

$$\frac{\partial^2 u}{\partial t^2} = \gamma^2 \Delta_{ND} u - 2\sigma_0 \frac{\partial u}{\partial t}, \tag{1}$$

with spatial variables scaled to object dimensions, where u is a membrane displacement, γ is a scaled wave velocity, σ_0 controls decay time T_{60}:

$$T_{60} = \frac{6\ln(10)}{\sigma_0}, \tag{2}$$

and $\Delta_{ND}u$ is a generalised N-dimensional Laplacian with x_i being subsequent spatial coordinates:

$$\Delta_{ND}u = \sum_{i=1}^{N} \frac{\partial^2 u}{\partial x_i^2}. \tag{3}$$

Hyper-membrane is clamped on all of its rectangular boundaries, which imposes the Dirichlet boundary conditions. In order to produce percussive sound a membrane has to be struck. The most basic way to simulate it is to impose initial conditions with spatial distribution such as the 'raised cosine' [2], which in N dimensions assumes the following form:

$$c_{rc}(\mathbf{x}) = \begin{cases} \frac{c_0}{2}\left(1 + \cos\left(\frac{\pi r_{ND}}{r_{hw}}\right)\right) & r_{ND} \le r_{hw} \\ 0 & r_{ND} > r_{hw} \end{cases}, \tag{4}$$

where c_0 is a peak displacement, its centre is given by $\mathbf{x}$, r_{hw} controls width of the distribution, and r_{ND} is an Euclidean distance in N-dimensional space from a peak location.

In the actual sound synthesizer described in this paper Eq. (1) is approximated with an explicit differential scheme, as given in Ref. [1]. Laplacian from Eq. (3) is approximated using an operator with adjacent neighbours only, without diagonals. Therefore calculation assumes a form of recursion where future solution in a particular grid point is calculated on the basis of solutions in $(2N+1)$ points from a current time step (central and nearest neighbours along N axes), and one (central) from a previous time step. Two coefficients are calculated for both central points, and one common for the neighbours.

2.3. Other models

Principles discussed in Sects. 2.1 and 2.2 can be applied to design other models of infeasible instruments. One can design hyper-plates by choosing a different equation instead of Eq. (1) to introduce stiffness – e.g. on a basis of the Kirchhoff model [17], and by changing boundary conditions. But even the same model may be extended by implementing more sophisticated models of membrane excitation, or by switching coordinate system to a different one, such as spherical, to design hyperspheres.

A very interesting case of an infeasible instrument can be designed on a basis of simple two-dimensional membrane by applying boundaries with a loop. Even though some loops may have physical representations, combining different loops on various boundaries is close to impossible in real objects. Assuming two dimensions p and q, grid sizes N_p and N_q, and grid point coordinates given by (l_p, l_q), the basic bi-directional loop along p requires to enable the following substitution in the FD grid:

$$\begin{matrix} u_{(-1,l_q)} \triangleq u_{(N_p-1,l_q)} \\ u_{(N_p,l_q)} \triangleq u_{(0,l_q)} \end{matrix}. \tag{5}$$

A perpendicular loop is applied as follows:

$$\begin{matrix} u_{(-1,l_q)} \triangleq u_{(l_q,N_q-1)} \\ u_{(l_p,N_q)} \triangleq u_{(0,l_p)} \end{matrix}. \tag{6}$$

Applying only one substitution from the pairs above will change a bi-directional loop into one-directional, even less feasible in a real object. One can also twist coordinate l_q while looping along p, which in a two-dimensional grid produces a vibrating Möbius strip. Spectra of such looped membranes are presented and discussed in Ref. [1].

Finally, all the objects mentioned can be supplemented with another infeasible property: an evolution of normally constant characteristic. It can be applied to material parameters, an instrument shape, or a fast relative movement of an instrument and a readout point [1].

3. Sound synthesizer

3.1. Implementation assumptions

Models of infeasible instruments presented in Sect. 2 have been previously proposed in Ref. [1]. They were initially implemented and tested to evaluate properties of sounds they are able to produce. However, these implementations were not designed to calculate output signal in real time, thus they were not fully

functional performance synthesizers, even though propositions regarding such implementations were provided. In this study a functional synthesizer has been implemented using a numerical model and directives provided in Ref. [1].

A four-dimensional hyper-membrane, as given in Eqs. (1)-(4), has been chosen for implementation. It was assumed that the instrument will be used as a percussion, therefore user needs to be able to control:

- exact moment of excitation (in order to perform rhythmic patterns),
- location of excitation on the membrane in all four dimensions simultaneously (to control sound timbre as in traditional membranophones),
- width of excitation spatial distribution (to further control sound timbre, as by switching drum sticks)
- strike velocity (to control musical dynamics).

It was to be established if the synthesizer can actually operate in real time, and what is the current limit of simulation parameters, assuming operation on a personal computer.

3.2. Hardware

The synthesizer requires two hardware elements to operate:

- a sound-enabled personal computer to perform numerical simulations and to produce a sound signal,
- a controller able to trigger membrane excitation, and continuously control location of an excitation along four axes simultaneously.

Control requirements can be met by more sophisticated MIDI controllers, e.g. with at least four sliders. However, quicker changes can be attained using a gamepad with two analogue thumbsticks, each one continuously controlling two axes (Fig. 1). A standard gamepad is equipped with twelve binary buttons that can trigger synthesizer events, or can be combined with thumbsticks movement to add more layers of continuous control.

Figure 1. Gamepad used as a control device.

3.3. Design

An implementation design of the synthesizer follows guidelines given in Ref. [1], and is similar to a design applied earlier in a synthesizer based on a model of normal, two-dimensional membrane, as described in Ref. [18]. It consists of two programs operating in parallel, and communicating through a network (Fig. 2). One is responsible for handling and interpreting events generated by the physical controller, here – a gamepad. The other performs numerical simulation and during this process generates an audio signal.

The controller program is actually a patch for the PureData graphical audio programming environment (Fig. 3). PureData is able to handle events from a broad range of different controllers, including MIDI controllers, computer vision controllers, gamepads, and other human interface devices (HUDs). It sends controller data using a network and the UDP protocol. This allows to physically separate the controller program from the simulation program, e.g. for a more convenient stage performance.

An FD simulation has a parallel nature – in theory all grid points in the same time step could be updated simultaneously. In order to exploit this possibility, the actual simulation is performed using the OpenCL [19] and a parallel processing device such as a graphics processing unit (GPU) or a multi-core central processing unit (CPU). The main simulation program itself is written in the C language for performance reasons, and it invokes the OpenCL kernel for parallel parts of the algorithm, that is for updating state of grid points. The OpenCL kernel is executed once for every time step, which is a performance limitation, but an efficient way to update a whole grid at the same time, using less processing elements than grid points, is yet to be determined due to severe limitations on communication between processing elements in the OpenCL. In a

current implementation the OpenCL kernel is executed by the main program in sequence, which synchronizes grid points in each time step, for a given number of time steps, to fill an audio buffer of predefined size. At the same time the program listens and collects control commands sent by the PureData patch to update the model before calculating next audio buffer.

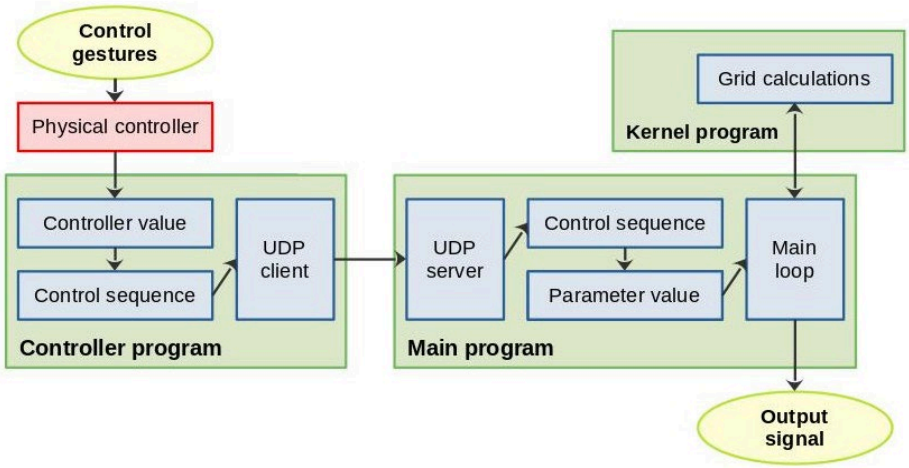

Figure 2. Outline of an implementation design of the synthesizer.

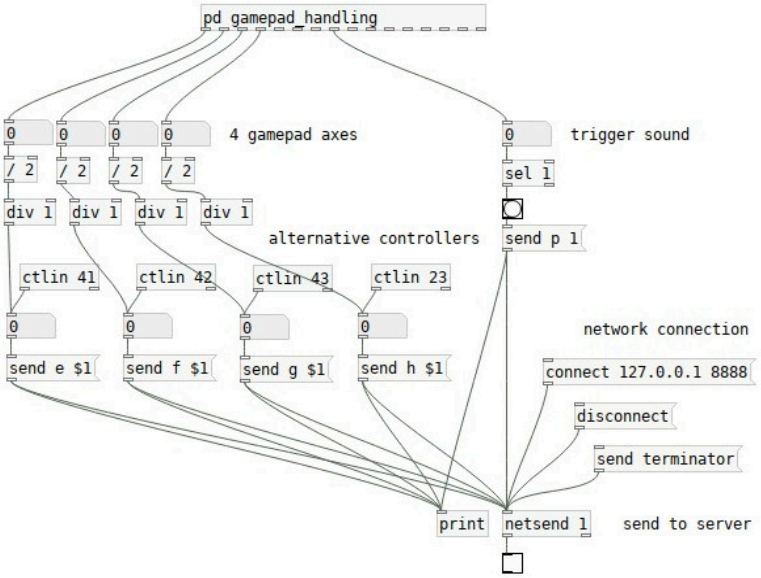

Figure 3. The controller program (main window) as a PureData patch.

In current version the parameters, including relative proportions of lengths of membrane sides, initial excitation size, velocity, and position, sampling frequency, and audio buffer length, or their initial values, if the value can be controlled by a user, are set through a configuration file.

4. Evaluation of operation

To determine if the synthesizer is able to perform in real time it was tested using a personal computer equipped with the AMD Ryzen 3700X CPU (8 cores, 16 threads), 32 GB of RAM memory, and the Nvidia Geforce 2070 Super GPU for the OpenCL calculations. Both programs, control and simulation, were running simultaneously on the same computer.

During tests it was established that a fluent operation, without dropped audio buffers (Fig. 4), was possible for a relatively low sampling frequency of 12 kHz with an audio buffer length set to at least 256 samples. In this case the stability condition determined a grid size of 28x20x17x15 elements (for a γ value

of 250.0). Operation with 16 kHz resulted in some dropped buffers, but it might be possible to use this frequency even with current hardware after applying some program optimisations.

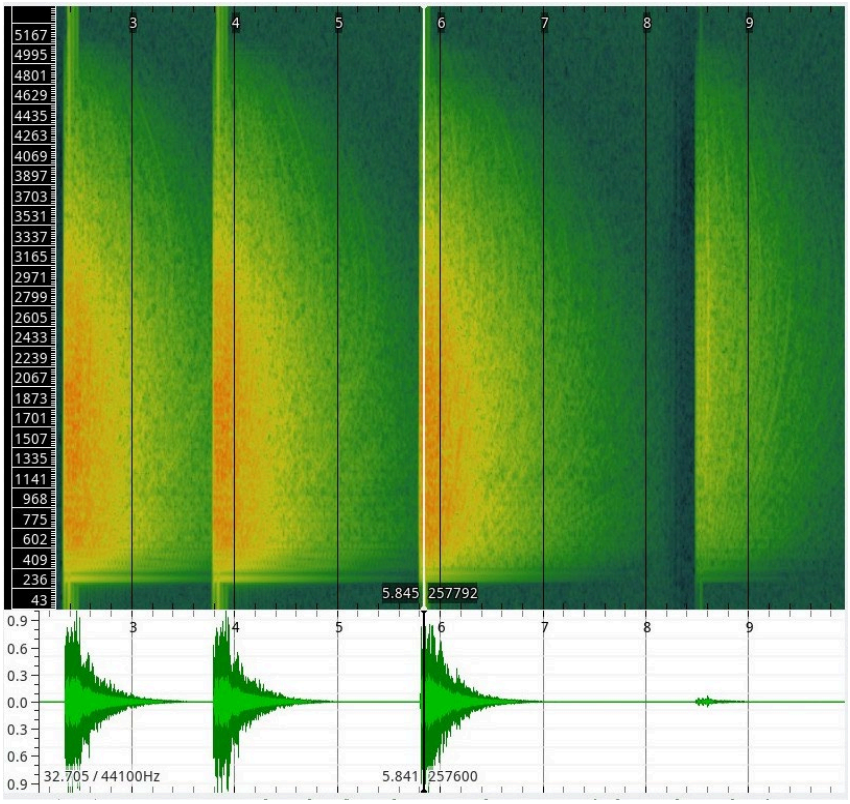

Figure 4. Waveform and spectrogram of a signal fragment recorded from the synthesizer.

5. Conclusions

Simulation of infeasible instruments is a technique of sound synthesis aimed at artistic sound experiments. It allows to hear and play instruments that cannot exist in reality. The scope of this paper was to present a possible implementation of such synthesizer, based on concepts and algorithms from previous works, and to determine if it was able to operate in real time. In this particular case it was a model of four-dimensional membrane.

The study carried out has determined that such operation is possible using a contemporary personal computer, although with a sampling frequency limited to 12 kHz. However, even with such limit the simulator is functional and produces sounds that may be considered interesting. Further optimisations in a parallel execution the algorithm shall allow to move the limit of sampling frequency towards higher values.

Concluding, it is interesting to observe how numerical simulations can be applied to explore areas of sounds partially unknown, and partially familiar, due to roots in some existing objects. Presented synthesizer design can be relatively easily adapted for other infeasible instruments mentioned in this paper, or for some combinations of their features.

Acknowledgments

The article was published as part of the research subsidy 16.16.130.942 of the Department of Mechanics and Vibroacoustics AGH-UST Krakow.

Additional information

The author declares: no competing financial interests and that all material taken from other sources (including their own published works) is clearly cited and that appropriate permits are obtained.

References

1. M. Pluta; Sound synthesis for music reproduction and performance; Wydawnictwa AGH: Krakow, Poland, 2019.
2. S. Bilbao; Numerical Sound Synthesis: Finite Difference Schemes and Simulation in Musical Acoustics; John Wiley Sons: New York, USA, 2009.
3. S. Bilbao, J. Perry, P. Graham, A. Gray, K. Kavoussanakis, G. Delap, T. Mudd, G. Sassoon, T. Wishart, S. Young; The NESS Project: Large Scale Physical Modeling Synthesis, Parallel Computing, and Musical Experimentation; Computer Music Journal 2020, 43(2-3), 31–47.
4. J.O. Smith III; Music applications of digital waveguides; Technical report, Department of Music, Stanford University, 1987.
5. J.O. Smith III; Physical Modeling Using Digital Waveguides; Computer Music Journal 1992, 16(4), 74–91.
6. J.O. Smith III; Acoustic modeling using digital waveguides; In: Musical Signal Processing; C. Roads, S.T. Pope, G. De Poli, A. Piccialli, Eds.; Swets & Zeitlinger: Downington, USA, 1997, 221–264.
7. J.M. Adrien, E. Ducasse; Dynamic Modeling of Vibrating Structures for Sound Synthesis, Modal Synthesis; In: Proceedings of the AES 7th International Conference: Audio in Digital Times, Toronto, Canada, May 14-17, 1989; Audio Engineering Society: New York, USA, 1990, 291–299.
8. J.D. Morrison, J.-M. Adrien; MOSAIC: A Framework for Modal Synthesis; Computer Music Journal 1993, 17(1), 45–56.
9. G. Eckel, F. Iovino, R. Causseé; Sound synthesis by physical modelling with Modalys; In: Proceedings of ISMA 1995, International Symposium on Musical Acoustics, Dourdan, France, July 2-6, 1995; 479–482.
10. C. Bruyns; Modal Synthesis for Arbitrarily Shaped Objects; Computer Music Journal 2006, 30(3), 22–37.
11. A. Gołaś, R. Filipek; Digital Synthesis of Sound Generated by Tibetan Bowls and Bells; Archives of Acoustics 2016, 41(1), 139–150.
12. C. Cadoz, A. Luciani, J.L. Florens, C. Roads, F. Chadabe; Responsive Input Devices and Sound Synthesis by Simulation of Instrumental Mechanisms: The CORDIS System; Computer Music Journal 1983, 8(3), 60–73.
13. C. Roads; The Computer Music Tutorial; The MIT Press: Massachusetts, USA, 1996.
14. P. Djoharian; Shape and Material Design in Physical Modeling Sound Synthesis; In: Proceedings of the 2000 International Computer Music Conference (ICMC), Berlin, Germany, August 27 – September 1, 2000; Michigan Publishing: Ann Arbor, USA, 2000, 38–45.
15. J. Leonard, C. Cadoz; Physical Modelling Concepts for a Collection of Multisensory Virtual Musical Instruments; In: Proceedings of the International Conference on New Interfaces for Musical Expression (NIME'15), Baton Rouge, USA, May 31 – June 3, 2015; Louisiana State University: Baton Rouge, USA, 2015, 150–155.
16. R. Courant, K. Friedrichs, and H. Lewy; Über die partiellen Differenzengleichungen der mathematischen Physik; Mathematische Annalen 1928, 100, 32–74.
17. P.M. Morse, K.U. Ingard; Theoretical Acoustics; Princeton University Press: Princeton, USA, 1987.
18. A. Czerwiński, D. Grzybek, P. Krauze, J. Łuczko, K. Michalczyk, P. Orkisz, M. Pluta, L. Radziszewski, M. Saga, J. Snamina; Synteza dźwięku z wykorzystaniem procesora graficznego, sterowana przy użyciu protokołu MIDI; In: Wybrane zagadnienia układów redukcji drgań i hałasu; Katedra Automatyzacji Procesów AGH: Kraków, Poland, 2015, 40–52.
19. A. Munshi, B. Gaster, T.G. Mattson, J. Fung, D. Ginsburg; OpenCL Programming Guide; Addison-Wesley Professional: Boston, USA, 2011.