Large Scale Physical Modeling Sound Synthesis

Large Scale Physical Modeling Sound Synthesis

Stefan Bilbao, Brian Hamilton,
Alberto Torin and Craig Webb
Acoustics Group, University of Edinburgh
sbilbao@staffmail.ed.ac.uk

Paul Graham, Alan Gray,
Kostas Kavoussanakis and James Perry
EPCC, University of Edinburgh

ABSTRACT

Sound synthesis based on physical models of musical instruments is, ultimately, an exercise in numerical simulation. As such, for complex systems of the type seen in musical acoustics, simulation can be a computationally costly undertaking, particularly if simplifying hypotheses, such as those of traveling wave or mode decompositions are not employed. In this paper, large scale time stepping methods, such as the finite difference time domain and finite volume time domain methods are explored for a variety of systems of interest in musical acoustics, including brass instruments, percussion instruments based on thin plate and shell vibration, and also their embeddings in 3D acoustic spaces. Attention is paid here to implementation issues, particularly on parallel hardware, which is well-suited to time stepping methods operating over regular grids. Sound examples are presented.

1. INTRODUCTION

Physical modeling sound synthesis has been approached in a variety of ways; perhaps the best known methods are the lumped mass spring network methodology, developed by Cadoz and associates [1], modal synthesis, developed at IRCAM [2] and digital waveguide methods developed by Smith at CCRMA [3], and subsequently greatly extended [4, 5, 6]. A direct comparison of these methods is difficult (but possible [7])—all possess distinct advantages: lumped network methods allow an extreme degree of control over the system, to the level of individual masses and springs; modal methods offer the possibility of exact solutions when modal data is available in closed form, or easily computed, and waveguides are extremely efficient when the system under consideration behaves (nearly) as the 1D wave equation, which is roughly true for a variety of systems of interest in musical acoustics.

Direct numerical simulation techniques, essentially time stepping methods such as the finite difference time domain method (FDTD) [8], or finite volume time domain method [9], though mainly used in other mainstream domains can also be applied to the problem of sound synthesis; indeed, they were proposed as long ago as 1969 by Ruiz [10], in the case of the vibrating string, and, in the scattering context by Kelly and Lochbaum [11] in the case of the vocal tract, and have been applied to a variety of problems, particularly in the musical acoustics setting, first by Chaigne and associates [12, 13, 14], and then others for synthesis [15]. Such brute force methods are often more computationally costly than the other techniques mentioned above; and yet, one advantage they possess is generality, in the sense that one may approach a great variety of systems, without the need for considering simplifying hypotheses (such as, e.g., linearity, or the availability of a modal decomposition or efficient traveling wave decomposition).

Until recently, it has been infeasible, computationally, to approach very large scale physical modeling synthesis using time stepping methods; however, the emergence into the mainstream of parallel hardware such as, e.g., general purpose graphical processing units (GPGPUs), has allowed the exploration of more complex systems. A project currently under way at the University of Edinburgh (NESS) is an exploratory attempt at producing synthetic sound for a variety of model systems, ultimately in 3D. Though not real time, computation time is becoming reasonable for certain systems; for other, particularly large systems (such as, e.g., the emulation of room acoustics), it is only now becoming possible to perform such simulations at all at audio rates, and on relatively cheap commerical hardware. GPGPUs offer one approach to optimisation among many,; whilst it would be feasible to use large-scale grid computing to run such simulations, the emphasis here is on hardware that is generally available for desktop computing. The overall aim of the project is to develop systems which can be used by musicians and composers, rather than to run simulations on a supercomputing machine.

There are many complications in designing such systems. Some are the usual difficulties in working with time stepping methods, such as, e.g., determining sufficient conditions for numerical stability, particularly under nonlinear conditions (the case of most interest in musical sound synthesis), or determining appropriate numerical boundary conditions when the geometry of the object under consideration is not simple. Others involve a reexamination of the properties of general numerical methods when audio synthesis is the goal, leading to various algorithmic constraints. Finally the question of parallelizability must be addressed if one is to eventually make the most of parallel hardware such as GPGPUs. There has been a good deal of recent work in modeling of room acoustics on GPGPU [16, 17, 18], but sound synthesis, both through physical

modeling [19] and other methods [20] has also been approached.

In this article, the emphasis is on design and implementation issues for FDTD methods for sound synthesis, rather than on musical acoustics. To this end, a simple family of model problems is introduced in Section 2. A brief overview of simple time stepping methods is given in Section 3, followed by a discussion of algorithm design issues, geared towards audio applications, in Section 4, and further parallelization issues on GPU in Section 5. Simulation results are presented in Section 6.

2. MODEL SYSTEMS

2.1 Second Order Systems

As a representative system, consider an object for which the dynamics are described by a partial differential equation of the form

$$\frac{\partial^2 U}{\partial t^2} = F \tag{1}$$

Here, t is a time variable, and $U = U(t, \mathbf{x})$ is the variable of interest to be solved for, such as a displacement of a bar, or plate, or pressure in an acoustic tube, and $\mathbf{x}$ is a spatial coordinate in d dimensions (normally $d = 1, 2$ or 3); generally the object is defined over a region $\mathbf{x} \in \mathcal{V} \subset \mathbb{R}^d$. F generally depends on U and its spatial derivatives (usually even) or temporal derivatives, or their combinations.

Such a model equation, though extremely simple, serves as a good first approximation to various systems of interest in musical acoustics. For example, consider the choices of F of

$$F = c^2 \nabla^2 U \tag{2}$$

$$F = -\kappa^2 \nabla^2 \nabla^2 U \tag{3}$$

$$F = \frac{c^2}{S(x)} \frac{\partial}{\partial x} \left(S(x) U \right) \tag{4}$$

$$F = c^2 \left(1 + \alpha \int_{\mathcal{V}} \left(\frac{\partial U}{\partial x} \right)^2 dx \right) \frac{\partial^2 U}{\partial x^2} \tag{5}$$

Here, c, κ and α are constants, and ∇^2 is the d-dimensional Laplacian operator, defined as

$$\nabla^2 = \sum_{\eta=1}^{d} \frac{\partial^2}{\partial x_\eta^2} \tag{6}$$

Equation 2 is the wave equation; when $d = 1$, it serves as an approximation to the vibration of a uniform string, where U is transverse displacement, and to the dynamics of a uniform acoustic tube, where U is pressure. When $d = 2$, it corresponds to the vibration of an ideal membrane, where U is transverse displacement, and when $d = 3$, it approximates wave propagation in an acoustic enclosure, where U is a variable such as a pressure or velocity potential.

Equation 3 is a linear model of vibration of a thin uniform bar ($d = 1$) or plate ($d = 2$), where U is transverse displacement.

Equation 4 is sometimes referred to as Webster's equation [21], and is a lossless 1D model of wave propagation, at speed c, in a tube of variable cross section $S(x)$, where U represents pressure or velocity potential.

Equation 5 is perhaps the simplest possible distributed nonlinearity in musical acoustics, sometimes referred to as the Kirchoff-Carrier equation [22, 23], and is used to model tension modulation effects [24] in strings, where again, U represents transverse displacement.

The model systems presented above are extremely simple, and in fact much too simple for good quality synthesis. For brevity, various features have been neglected here:

All the systems above are lossless; real systems possess a variety of internal dissipation mechanisms, such as thermal/viscous effects, and also transfer energy to their surroundings through radiation. In fully 3D synthesis models, radiation is neatly handled through direct coupling to the acoustic field; modeling of thermal/viscous effects, however, is much more involved using time stepping methods. See Section 6.1 for more on this topic.

Boundary conditions have not been specified here; in some cases in musical acoustics, these can be trivial (as in, e.g., the case of an ideal rigidly terminated string), but in others can require extreme care as in, e.g., free edges of thin structures such as cymbals or gongs, radiation impedance conditions in 1D models of wind instruments, and absorbing boundary conditions in 3D acoustic simulations.

Equation 1 represents the behaviour of an unforced system; that is to say, it is lacking, as yet, an excitation term. For most systems of interest, the excitation is nonlinearly dependent on U, but acts at a single location (or very small region) on the object; nonlinearity in F itself has a much greater impact on analysis and numerical design for the system, and is sometimes referred to as a distributed nonlinearity.

Not all systems of interest take the above form—one example is the system describing nonlinear wave propagation in an acoustic tube, which is most naturally written as a first order system [9]. In a modular setting, when dealing with systems in contact with the acoustic field, and also in more elaborate models of stiff systems such as bars and plates the variable U is coupled to other variables—alternatively, in such cases, one could interpret U as a vector variable.

3. TIME STEPPING METHODS

The first approximation necessary is discretization in time. A simple choice, and a natural one in audio applications, is an approximation at equally spaced intervals of k seconds ($F_s = 1/k$ is the sample rate), and thus in discrete time, $u^n(\mathbf{x})$ represents an approximation to $U(nk, \mathbf{x})$, where n is an integer (the time index).

Considering first the second time derivative operator, a multitude of approximation methods are available, and in particular multistep methods such as Runge Kutta, Adams Bashfort, etc. [25]. Given that in large simulations in parallel hardware, memory use may be a bottleneck, depending on accuracy requirement it may be useful to make use of the simplest possible approximation:

$$\left. \frac{\partial^2 U}{\partial t^2} \right|_{t=nk} \quad \Longrightarrow \quad \frac{1}{k^2} \left(u^{n+1} - 2u^n + u^{n-1} \right) \tag{7}$$

Such an approximation, though only second order accurate [26] is minimal in terms of memory usage. (For some systems, particularly if there are long memory effects associated with viscothermal losses requiring more memory, then the need for such simple approximations is less urgent—see Section 6.1.)

The character of the resulting algorithm depends greatly on the discretization of F. If the simple approximation $F^n = f(U(nk, \mathbf{x}))$ is used, then an update for 1 will be of the form

$$u^{n+1} = 2u^n - u^{n-1} + k^2 f^n \tag{8}$$

and, after suitable discretization over a grid (see Section 3.1), will be fully explicit: at each time step, values of u^{n+1} at the next time step may be computed directly from previously calculated values of u and f at time steps n and $n-1$.

In some cases, however, for reasons having to do with stability [26], and also in reducing artifacts resulting from numerical dispersion [15], it may be preferable to use a different approximation, such as, e.g., $\bar{f}^n = \frac{1}{2}\left(f^{n+1} + f^{n-1}\right)$; in this case, the update for 1 becomes

$$u^{n+1} = 2u^n - u^{n-1} + \frac{k^2}{2}\left(f^{n+1} + f^{n-1}\right) \tag{9}$$

which is implicit—u^{n+1} and f^{n+1} (which depends on u^{n+1}) must be computed together at each time step. If the system under consideration is linear, then this amounts to a linear system solution, which often complicates implementation in parallel hardware. If the system is nonlinear, then existence and uniqueness issues in the update above may appear. For more on issues related to implicit methods, see Section 3.2.

3.1 Grids

In the FDTD setting, numerical solutions are often represented over uniform grids—see Figure 1, showing simple Cartesian grids in 1D, 2D and 3D. Working over such regular grids has, of course, strengths and weaknesses. The most significant benefit is in terms of parallelizability, particularly when the problem under consideration is uniform over space, as the action of a given approximation to a differential operator is the same at each point in the domain. Such is the case, for example, for membranes and plates of constant thickness, in tubes of uniform cross section, and for 3D acoustic wave propagation. Difficulties emerge, however, when treating irregular boundaries, both in terms of analysis, as well as in parallel implementation. The former difficulty, however, can be addressed by an appeal to methods over unstructured grids, such as, e.g., finite volume techniques [9]; see [27] for more on this topic in the case of the 3D wave equation and room acoustics applications.

Supposing operation over a regular Cartesian grid, of spacing h between adjacent grid points, then grid functions $\hat{u}$ are fully discrete approximations to the solution U. For example, the grid functions $\hat{u}_l^n$, $\hat{u}_{l,m}^n$ and $\hat{u}_{l,m,p}^n$ are approximations to the solution $U(t, \mathbf{x})$ at time $t = nk$, and at locations $\mathbf{x} = lh$, $\mathbf{x} = [l\,m]h$ and $\mathbf{x} = [l\,m\,p]h$ in 1D, 2D and 3D respectively, for integer l, m and p.

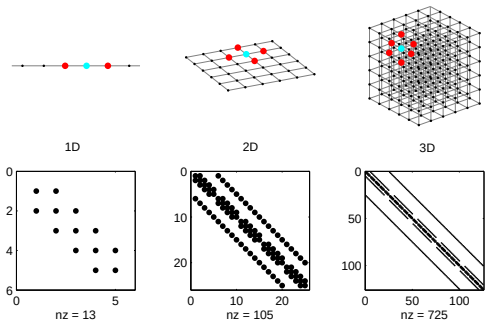

Figure 1. Top row: Cartesian grids in 1D, 2D and 3D. At a given grid point (in blue), the family of neighboring points over which a simple approximation to the Laplacian operates is indicated in red. Bottom row: sparsity plots for Laplacian approximations, when the operation is represented as a matrix multiplication.

Difference operations approximating differential operators are straightforward to obtain. Consider the Laplacian operator, which plays a central role in many physical models. The simplest approximations in 1D, 2D and 3D, written here as $\delta_{\nabla^2}^{(1D)}$, $\delta_{\nabla^2}^{(2D)}$ and $\delta_{\nabla^2}^{(3D)}$, respectively, are

$$\delta_{\nabla^2}^{(1D)} u_l^n = \frac{1}{h^2}\left(u_{l+1}^n - 2u_l^n + u_{l-1}^n\right) \tag{10}$$

$$\delta_{\nabla^2}^{(2D)} u_{l,m}^n = \frac{1}{h^2}\Big(u_{l+1,m}^n + u_{l-1,m}^n + u_{l,m+1}^n + u_{l,m-1}^n - 4u_{l,m}^n\Big) \tag{11}$$

$$\delta_{\nabla^2}^{(3D)} u_{l,m,p}^n = \frac{1}{h^2}\Big(u_{l+1,m,p}^n + u_{l-1,m,p}^n + u_{l,m+1,p}^n + u_{l,m-1,p}^n + u_{l,m,p+1}^n + u_{l,m,p-1}^n - 6u_{l,m,p}^n\Big) \tag{12}$$

Such approximations employ nearest neighbours only on a regular grid—see Figure 1 for a graphical representation of the region of operation of such operators (or stencils) over Cartesian grids.

3.2 Sparse Vector Matrix Representations and Recursions

For computing purposes, is often convenient to represent the grid functions $hatu$, which are multidimensional arrays, as vectors $\hat{\mathbf{u}}$—see Figure 2, showing concatenation of columns of a 2D array into a vector. In this formalism, difference operators are represented as matrix multiplications, where, due to the local character of finite difference operations, the matrices are sparse. See the bottom row of Figure 1, showing sparsity plots of matrix representations of the Laplacian operator, in 1D, 2D and 3D.

Using this formalism, in many cases, it is then possible to write recursions such as (8) and (9) in vector matrix form as

$$\mathbf{A}\hat{\mathbf{u}}^{n+1} = \mathbf{B}\hat{\mathbf{u}}^n + \mathbf{C}\hat{\mathbf{u}}^{n-1} \tag{13}$$

where here, $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are update matrices incorporating the effects of various finite difference approximations over the grid, and are generally sparse.

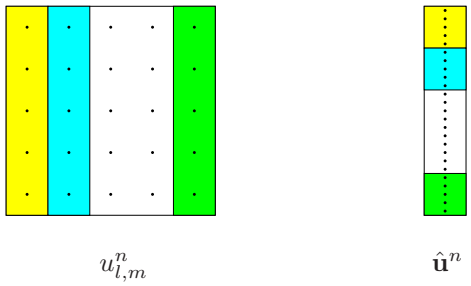

Figure 2. Reorganisation of a 2D array $u^n_{l,m}$ as a vector, by concatenation of columns.

Various special cases emerge at this point.

If the scheme is explicit, as in, say, the case of (8), then $\mathbf{A}$ is simply the identity matrix, and thus the solution may be updated solely through sparse matrix multiplication operations (of $\mathbf{B}$ and $\mathbf{C}$ in (13)). This is the ideal case in a parallel realization.

If the scheme is implicit, but linear, then $\mathbf{A}$ will not be the identity, and a linear system solution will be required in the loop in order to solve for $\hat{\mathbf{u}}^{n+1}$. However, as it is constant, it is possible to some perform preconditioning once, offline.

If the scheme is implicit, and also nonlinear, then it may not be possible to write a recursion such as the above—in some cases, and particularly when the nonlinearity is of a simple form (such as, e.g., cubic, as occurs frequently in models of vibration of strings and plates, and as exhibited in (5)), then one may arrive at such a form—but generally $\mathbf{A}$ is dependent on previously computed values of the solution $\hat{\mathbf{u}}^n$, and thus must be constructed anew at each time step. This is certainly the most challenging case in a parallel realization.

4. ALGORITHM DESIGN ISSUES

4.1 Linear System Solutions

For implicit methods, as described above, linear system solutions are necessary in updating the state of the system. For methods defined locally over a grid, such as FDTD, such matrices are sparse and possess a banded structure (see Figure 1).

Many efficient methods are available in approaching the solution of sparse banded systems; among the best known are methods such as the Thomas algorithm (and extensions) [26]; but such methods generally require diagonal dominance, which is not always the case for matrix representations of FDTD schemes. Also, they are inherently serial—one must proceed, step by step, along the bands of the matrix in order to arrive at a solution. For an efficient realization in parallel hardware, parallelizable methods must clearly be employed. There are many such methods available—particularly well-suited to sparse systems are iterative methods such as the conjugate gradient family, employing a sparse preconditioner (such as incomplete Cholesky factorization). The problem of determining this preconditioner (possibly anew at each time step) must be weighed against the computational cost of determining the preconditioner.

In other cases, such as, e.g., those involving interpolation between distinct grids (as in the case of systems coupled to the acoustic field), this banded structure may be lost; in compensation, however, the system to be solved may be strongly diagonally dominant, allowing the use of very simple iterative methods (such as, e.g., Gauss-Seidel [26]). For an example of such interpolation between grids in matrix form, see, e.g., [19].

4.2 Stability

For explicit schemes, which are of most interest in parallel implementation, stability conditions for schemes for the model system 1 are best framed in terms of a lower bound on the grid spacing h in terms of k, the time step (which, in audio applications, is normally set a priori): in other words,

$$h \geq h_{min}(k) \tag{14}$$

One way of arriving at such conditions, for linear problems, is through the use of frequency domain techniques, or von Neumann analysis [26]; such methods, however, do not apply directly to nonlinear systems, nor do they allow the determination of sufficient stability conditions when boundary are included. As an alternative, methods based on discrete energy conservation/dissipation are employed [15, 28], which do allow such stability conditions to be determined under very general conditions.

4.3 Bandwidth Limitation

Given the lower bound on h in (14), it may be tempting to choose a value of h which is larger than the minimum, in the interest of reducing computational complexity; this, however, leads to severe limitations on output bandwidth—see Figure 3. Accompanying this is a phenomenon known as numerical dispersion, leading to a mistuning of modal frequencies; for this reason, in audio applications, it is generally best to choose the grid spacing as close to the stability condition as possible.

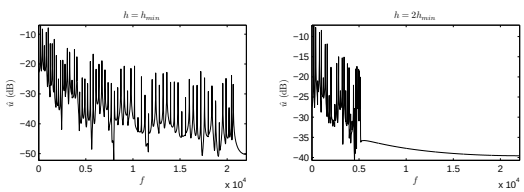

Figure 3. Spectrum of sound output from a simple stiff string physical model, at 44.1 kHz. Left: where the grid spacing is chosen at the minimum allowable value from the stability condition, and right, at twice the minimum value, illustrating a severe numerical cutoff at approximately 5 kHz.

4.4 Computational Complexity

Computational complexity, as expected, scales strongly with the dimension of the system; in particular, in 3D, computational costs for large volume simulations are extreme. See

Table 1 for typical computational costs for FDTD methods, in terms of memory required to hold the state, and floating point operations per second output, at a typical audio rate. Such costs motivate an examination of acceleration in parallel hardware, as discussed in the next section.

	Memory (B)	Flop/s
Tube, length 1 m	3.11×10^{3}	2.86×10^{7}
String, steel, length 1 m radius 1×10^{-3} m, tension 100 N	1.66×10^{4}	1.53×10^{8}
Plate, steel, area 1 m^3, thickness 5×10^{-4} m	3.46×10^{5}	1.08×10^{10}
Small enclosure (1 m^3)	1.00×10^{7}	1.67×10^{11}
Large room (10^4 m^3)	1.00×10^{11}	1.67×10^{15}

Table 1. Memory requirements, in B, and floating point operations per second output for several typical systems, for standard FDTD schemes operating at $F_s = 44.1$ kHz, and using double precision arithmetic.

5. IMPLEMENTATION ON GPU

In this section, the focus is on the use of GPGPUs, using Nvidia's CUDA architecture.

The starting point for the implementation of the time stepping schemes described above is generally a prototype code in a high level language such as MATLAB that is especially useful for dealing with the matrix structures and operations that arise. In order to move to a GPU implementation, the prototype models are rewritten first as serial C code, and then the time critical elements are threaded to run in parallel using the CUDA language (see Figure 4). Threads perform a kernel operation in a SIMD manner (Single Instruction Multiple Data) on the GPU device. These threads are grouped into blocks of up to 1024 threads, and then multiple blocks form the thread grid.

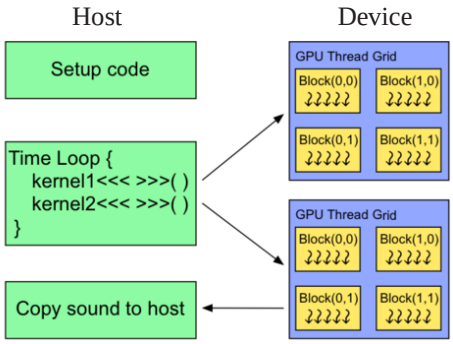

Figure 4. Code design for host and device

Following from the discussion in Section 3.2, from an implementation perspective, the systems can be grouped into four categories:

1. Uniform explicit schemes.
2. Non-uniform, but explicit, schemes.
3. Implicit schemes, with constant update matrices.
4. Implicit schemes, with update matrices constructed at each time step.

The first case, of uniform schemes, are explicit and where the update matrices $\mathbf{B}$ and $\mathbf{C}$ contain bands of constant coefficients. They are often of Toeplitz or block Toeplitz form. For such systems, the matrix form can be 'unrolled' such that grid points can be updated by a simple equation using scalar coefficients and neighbouring points. This greatly reduces the amount of memory access required, which is one of the main concerns for the GPU implementation. FDTD schemes generally suffer from a low compute-to-memory access ratio, and so efficiency is always limited by data transfer rather than peak computing performance. For even the simplest scheme there are still many possible approaches to thread design and various other optimisations, and therefore a certain amount of experimentation is usually required to find the most efficient CUDA solution.

The second case, where the update matrices are not uniform but the scheme is still explicit (such as, e.g., the case of schemes for Webster's equation (4)), requires either some storage system for holding the coefficients, or operating directly with sparse matrix objects. A suitable code library is required to handle sparse matrices in C code and perform basic linear algebra operations. These operations then need to be performed, and hopefully accelerated, on the GPU. Whilst some libraries are available for this purpose, such as Nvidia's cuSparse [29], custom-designed functions may well provide a more efficient solution.

The majority of the matrices involved are of multi-banded, or block banded form. Of the various sparse matrix formats available, the DIA (or diagonal) format is particularly suitable. Each band is set as a column in a table, with a small integer array to indicate the distance of each column away from the centre diagonal, as shown in Figure 5. This format

$$\begin{bmatrix} 1 & 2 & . & . & . \\ . & 7 & 8 & . & . \\ 11 & . & 13 & 14 & . \\ . & 17 & . & 19 & 20 \\ . & . & 23 & . & 25 \end{bmatrix} \longrightarrow \begin{array}{c} \begin{bmatrix} -2 & 0 & 1 \end{bmatrix} \\ \begin{bmatrix} . & 1 & 2 \\ . & 7 & 8 \\ 11 & 13 & 14 \\ 17 & 19 & 20 \\ 23 & 25 & . \end{bmatrix} \end{array}$$

Figure 5. Sparse matrix representation in DIA format.

provides an efficient representation, and a library of linear algebra functions is being developed that are optimised specifically for the systems that arise. Some elements, such as the interpolation between 2D and 3D systems, result in matrices that are not banded in structure. In this case a more general format, such as CSR, is still required.

The third and fourth categories of code are implicit, and thus require a solution to a system of linear equations at each time step of the simulation. This may use either a persistent, unchanging matrix, or may require the construction of a matrix system at each iteration. In prototyping it is simple enough to use MATLAB's backslash operator in this situation. Accelerating this in C code and on the GPU requires more complex libraries such as PETSc [30], or the development of custom functions based on iterative methods [31].

6. SYSTEMS AND SIMULATIONS

In this section, simulation results using FDTD methods are presented for several families of systems. Accompanying sound examples and video demonstrations appear on the NESS project website: `www.ness-music.eu`

6.1 Brass Instruments

As mentioned in Section 2, a starting point for brass instrument synthesis is a 1D model such as Webster's equation 4. In the lossless case, simple, provably stable numerical methods are available [15]; for good quality synthesis, however, various refinements are necessary. Chief among these is the modeling of viscothermal boundary layer losses in the tube, which is typically described in terms of input impedance [32, 33]; when translated to the time domain, fractional time derivative terms appear, which in FDTD requires recursions of order higher than two, and thus the memory requirement/operation count is increased. A second feature of interest involves user-controlled time variation (through, e.g., slides or valves); such time variation is handled locally within an FDTD scheme, requiring no additional precomputation (in contrast with, e.g., modal methods, which would require a recalculation of modal shapes and frequencies for every valve configuration). Finally, if nonlinear effects (due to shock formation in long cylindrical bore instruments such as the trombone [34]) are to be introduced, variants of Webster's equation are no longer suitable, and it is best to revert to a first order system, as is common in the maintream finite volume literature [9]; numerical stability is difficult to maintain under such conditions, without introducing artificial viscosity, which can impact negatively on sound quality.

6.2 Percussion: Nonlinear Plate and Shell Vibration

The high amplitude vibration of thin rigid structures such as plates and shells features strongly in many percussion instruments, including cymbals, gongs and tamtams, and also impacts to a lesser extent on the sound of drums. In general, for such instruments, under a striking excitation, there is a migration of energy from low frequencies to high, giving rise to crash like effects in cymbals, and slow swells in gongs—see Figure 6, showing a spectrogram of sound output for a typical nonlinear flat plate model. Linear models, such as that of, e.g., Kirchhoff Love [35] do not capture such effects, and thus nonlinear models are necessary—that of von Kármán [36] is probably the simplest to adequately render such effects. In the modal setting, the nonlinearity leads to the transfer of energy between modes—see Figure 7.

From an implementation perspective, the main difficulty is in performing linear system solutions in the time recursion; though sparse, due to the nonlinearity, the linear system to be solved must be constructed anew at each time step, as described in Section 3.2, and thus linear system solution techniques can become computationally quite heavy.

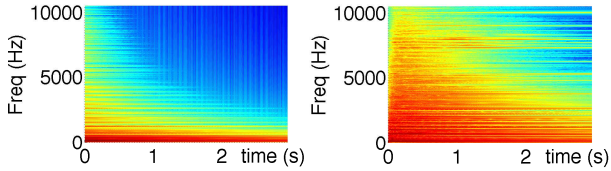

Figure 6. Spectrogram of sound output for a rectangular plate under linear conditions (left) and nonlinear conditions (right).

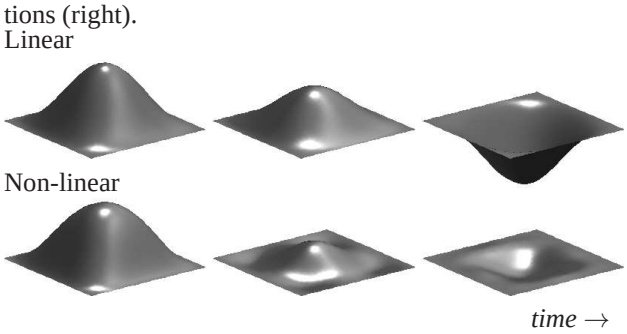

Figure 7. Linear (top) and nonlinear (bottom) time evolution of the displacement of a thin square plate, under simply supported boundary conditions. In both cases, the initial condition is set to the lowest linear mode of vibration.

6.3 Modular Environments

One of the main goals of many physical modeling sound synthesis paradigms (including the CORDIS environment [37], MOSAIC/Modalys [38], and BlockCompiler [39] is modular construction of new virtual instruments. The goal is no different in the present case of FDTD methods, where here, the canonical elements are distributed, possibly nonlinear objects such as bars, plates, strings, membranes and acoustic tubes, all represented over distinct grids, and coupled through a variety of connection types.

In Figure 8, at left, a plate based percussion instrument is shown, with connections (of mass/spring/damper type, as in CORDIS) indicated by blue lines. In Figure 8, at right, a modular instrument constructed from a set of acoustic tubes is shown—here, as discussed in Section 6.1, the configuration is time varying, allowing for half-valve effects [40].

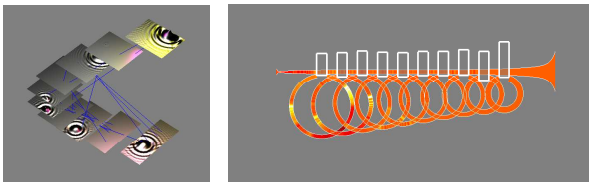

Figure 8. Left: Modular percussion instrument made up of a collection of thin plates, with connections indicated by blue lines. Right: Modular brass instrument made up of a collection of acoustic tubes.

6.4 3D Acoustic Spaces

Modeling of 3D acoustic spaces using FDTD, for room acoustics modeling and artificial reverberation applications

is a well-researched topic [41, 42, 43]. Acceleration using GPGPUs is of great importance in this case, given the computational scale of the problem.

Table 2 shows benchmark times for the standard rectilinear 3D FDTD scheme at various grid sizes; the simulation was computed for 44,100 samples at 44.1kHz. Serial C code was tested on an Intel Xeon E5-2620 with -O3 compiler optimisation, whilst the CUDA code was tested on an Nvidia Tesla K20. Both double (DP) and single (SP) precision floating-point arithmetic tests were performed.

Test	Serial C Intel	CUDA Tesla	Speedup
$1m^3$ DP	55 sec	8.7 sec	x6.3
$1m^3$ SP	55 sec	7.2 sec	x7.3
$50m^3$ DP	52.7 min	3.3 min	x16.0
$50m^3$ SP	51.6 min	2.0 min	x25.8
$500m^3$ DP	531.3 min	31.9 min	x16.6
$500m^3$ SP	528.7 min	18.9 min	x28.0

Table 2. Benchmarks for serial C code vs CUDA.

Beyond the question of raw acceleration, numerous other features in room acoustics required a detailed examination from a numerical perspective. One is air viscosity, leading to damping of wave propagation at high frequencies [44], and which is especially important to avoid unnatural ringing in computed solutions in large spaces. Another is that of stable boundary termination, especially over irregular geometries, and when realistic wall impedance conditions are taken into account. Finally, care must be taken when choosing the grid to be used in 3D space—see Figure 9 for two such choices. There are great variations in the effects of numerical dispersion depending on the grid—and also on GPU implementation.

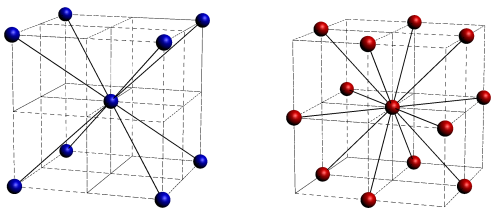

Figure 9. Computational grids in 3D: Left, body-centred cubic, and right, face-centred cubic.

6.5 Embeddings of Instruments in 3D

The ultimate goal of physical modeling is, perhaps, the full emulation of musical instruments in a 3D enclosure. New issues emerge here regarding the coupling between the acoustic field (described by a variant of the 3D wave equation 2, as detailed above) and the object at hand.

See Figure 10, illustrating the time evolution of the acoustic field surrounding a set of timpani drums, within a 3D enclosure.

7. CONCLUSIONS AND PERSPECTIVES

This paper is intended as an overview of the use of FDTD methods in sound synthesis, particularly as applied to large,

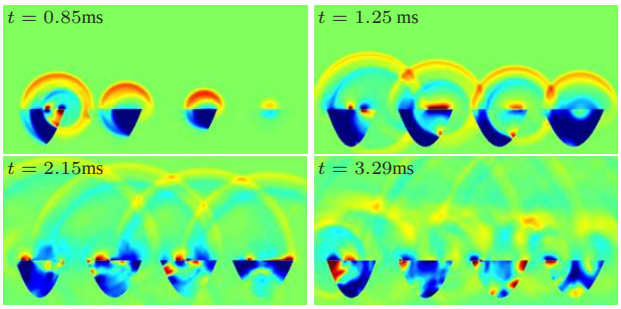

Figure 10. Snaphots of the time evolution of the acoustic field surrounding a set of four struck timpani drums.

real-world physical models, and ultimately in 3D. Such methods, which involve local or nearest neighbour updates, are a good match to implementation in parallel hardware. As one might expect, when a state space representation is employed, such methods reduce to large sparse matrix operations performed in a time loop—and in particular multiplications, and inversions (or linear system solutions). In general, it is the latter which poses the most problems in a parallel implementation, though due to matrix sparsity, specialized methods are available. For very large problems, when a linear system solution is unnecessary (as is the case for large 3D simulations), an implementation on GPU leads to great accelerations in computation time.

From an algorithm design point of view, time stepping methods represent, in some respects, a last resort—for sufficiently complex systems, involving distributed nonlinearities, and nontrivial couplings among disparate components (including the acoustic field itself), there does not appear to be any other avenue of approach. One great strength of such an approach not mentioned in the Introduction is that, ultimately, in 3D simulations, one has complete access to the acoustic field surrounding such as instrument—and thus complete control over spatialization of the resulting sound output. The difficulties, however, are many—great care must be taken at the design stage, compared with other methodologies, to ensure that numerical artifacts (such as, e.g., bandwidth limitation and numerical dispersion) do not impact negatively on sound output.

One issue which has not been broached here in any detail (and which is premature, given current compute times) is, as is usual in physical modeling synthesis, user control. Particularly in the setting of modular instrument constructions, which may be quite a bit more complex than real world acoustic instruments, finding a meaningful and parsimonious means of both designing and playing such instruments presents a daunting challenge.

Acknowledgments

This work was supported by the European Research Council, under grant StG-2011-279068-NESS.

8. REFERENCES

[1] C. Cadoz, A. Luciani, and J.-L. Florens, "Responsive input devices and sound synthesis by simulation of instrumental mechanisms," *Computer Music Journal*, vol. 8, no. 3, pp. 60–73, 1983.

[2] J.-M. Adrien, "The missing link: Modal synthesis," in *Representa-*

tions of Musical Signals, G. DePoli, A. Picialli, and C. Roads, Eds. Cambridge, Massachusetts: MIT Press, 1991, pp. 269–297.

[3] J. O. Smith III, "Physical modelling using digital waveguides," *Computer Music Journal*, vol. 16, no. 4, pp. 74–91, 1992.

[4] P. Cook, "Tbone: An interactive waveguide brass instrument synthesis workbench for the NeXT machine," in *Proceedings of the International Computer Music Conference*, Montreal, Canada, 1991, pp. 297–299.

[5] M. Karjalainen, V. Välimäki, and T. Tolonen, "Plucked-string synthesis: From the Karplus-Strong algorithm to digital waveguides and beyond," *Computer Music Journal*, vol. 22, no. 3, pp. 17–32, 1998.

[6] V. Välimäki, M. Laurson, and C. Erkut, "Commuted waveguide synthesis of the clavichord," *Computer Music Journal*, vol. 27, no. 1, pp. 71–82, 2003.

[7] V. Välimäki, J. Pakarinen, C. Erkut, and M. Karjalainen, "Discrete time modeling of musical instruments," *Reports on Progress in Physics*, vol. 69, pp. 1–78, 2006.

[8] A. Taflove, *Computational Electrodynamics*. Boston, Massachusetts: Artech House, 1995.

[9] R. Leveque, *Finite Volume Methods for Hyperbolic Problems*. Cambridge, UK: Campbridge University Press, 2002.

[10] P. Ruiz, "A technique for simulating the vibrations of strings with a digital computer," Master's thesis, University of Illinois, 1969.

[11] J. Kelly and C. Lochbaum, "Speech synthesis," in *Proceedings of the Fourth International Congress on Acoustics*, Copenhagen, Denmark, 1962, pp. 1–4, paper G42.

[12] A. Chaigne, "On the use of finite differences for musical synthesis. Application to plucked stringed instruments," *Journal d'Acoustique*, vol. 5, no. 2, pp. 181–211, 1992.

[13] A. Chaigne and A. Askenfelt, "Numerical simulations of struck strings. I. A physical model for a struck string using finite difference methods," *Journal of the Acoustical Society of America*, vol. 95, no. 2, pp. 1112–1118, 1994.

[14] L. Rhaouti, A. Chaigne, and P. Joly, "Time-domain modeling and numerical simulation of a kettledrum," *Journal of the Acoustical Society of America*, vol. 105, no. 6, pp. 3545–3562, 1999.

[15] S. Bilbao, *Numerical Sound Synthesis: Finite Difference Schemes and Simulation in Musical Acoustics*. Chichester, UK: John Wiley and Sons, 2009.

[16] A. Southern, D. Murphy, G. Campos, and P. Dias, "Finite difference room acoustic modelling on a general purpose graphics processing unit," in *Audio Engineering Society Convention 128*, 2010. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=15325

[17] R. Mehra, N. Raghuvanshi, L. Savioja, M. Lin, and D. Manocha, "An efficient GPU-based time domain solver for the acoustic wave equation," *Applied Acoustics*, vol. 73, no. 2, pp. 83–94, 2012.

[18] C. Webb and S. Bilbao, "Computing room acoustics with cuda - 3d fdtd schemes with boundary losses and viscosity," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2011, pp. 317–320.

[19] S. Bilbao and C. Webb, "Timpani drum synthesis in 3d on gpgpus," in *Proceedings of the 15th International Conference on Digital Audio Effects*, York, UK, September 2012.

[20] L. Savioja, V. Valimaki, and J. Smith, "Audio signal processing using graphics processing units," *J. Audio Eng. Soc.*, vol. 59, no. 1/2, Feb 2011.

[21] A. Webster, "Acoustical impedance, and the theory of horns and of the phonograph," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 5, no. 7, pp. 275–282, 1919.

[22] G. Kirchhoff, *Vorlesungen über Mechanik*. Leipzig: Tauber, 1883.

[23] G. Carrier, "On the nonlinear vibration problem of the elastic string," *Quarterly of Applied Mathematics*, vol. 3, pp. 157–165, 1945.

[24] T. Tolonen, V. Välimäki, and M. Karjalainen, "Modelling of tension modulation nonlinearity in plucked strings," *IEEE Transactions in Speech and Audio Processing*, vol. 8, pp. 300–310, 2000.

[25] R. S. A. Quarteroni and F. Saleri, *Numerical Mathematics*. Springer, 2007.

[26] J. Strikwerda, *Finite Difference Schemes and Partial Differential Equations*. Pacific Grove, California: Wadsworth and Brooks/Cole Advanced Books and Software, 1989.

[27] S. Bilbao, "Modeling of complex geometries and boundary conditions in finite difference/finite volume time domain room acoustics simulation," *IEEE Transactions on Audio Speech and Language Processing*, 2013, in press.

[28] B. Gustaffson, H.-O. Kreiss, and J. Oliger, *Time Dependent Problems and Difference Methods*. New York, New York: John Wiley and Sons, 1995.

[29] Nvidia, "CUSPARSE library," *CUDA toolkit documentation.[Online][Cited: 29th Mar 2013.] http://docs.nvidia.com/cuda/*, 2013.

[30] S. Balay, W. Gropp, L. McInnes, and B. Smith, "Efficient management of parallelism in object oriented numerical software libraries," in *Modern Software Tools in Scientific Computing*, E. Arge, A. M. Bruaset, and H. P. Langtangen, Eds. Birkhäuser Press, 1997, pp. 163–202.

[31] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003.

[32] D. Keefe, "Acoustical wave propagation in cylindrical ducts: Transmission line parameter approximations for isothermal and nonisothermal boundary conditions," *Journal of the Acoustical Society of America*, vol. 75, no. 1, pp. 58–62, 1984.

[33] R. Caussé, J. Kergomard, and X. Lurton, "Input impedance of brass musical instruments—comparison between experiment and numerical models," *Journal of the Acoustical Society of America*, vol. 75, no. 1, pp. 241–254, 1984.

[34] A. Hirschberg, J. Gilbert, R. Msallam, and A. Wijnands, "Shock waves in trombones," *Journal of the Acoustical Society of America*, vol. 99, no. 3, pp. 1754–1758, 1996.

[35] K. Graff, *Wave Motion in Elastic Solids*. New York, New York: Dover, 1975.

[36] A. Nayfeh and D. Mook, *Nonlinear Oscillations*. New York, New York: John Wiley and Sons, 1979.

[37] C. Cadoz, A. Luciani, and J.-L. Florens, "Cordis-anima: A modeling and simulation system for sound and image synthesis," *Computer Music Journal*, vol. 17, no. 1, pp. 19–29, 1993.

[38] D. Morrison and J.-M. Adrien, "Mosaic: A framework for modal synthesis," *Computer Music Journal*, vol. 17, no. 1, pp. 45–56, 1993.

[39] M. Karjalainen, "Block-compiler: Efficient simulation of acoustic and audio systems," presented at the 114th Audio Engineering Society Convention, Amsterdam, the Netherlands, May, 2003. Preprint 5756.

[40] S. Bilbao, "Modeling of brass instrument valves," in *Proceedings of the 14th International Conference on Digital Audio Effects*, Paris, France, September 2011.

[41] L. Savioja, T. Rinne, and T. Takala, "Simulation of room acoustics with a 3-D finite-difference mesh," in *Proceedings of the International Computer Music Conference*, Århus, Denmark, September 1994, pp. 463–466.

[42] D. Botteldooren, "Finite-difference time-domain simulation of low-frequency room acoustic problems," *Journal of the Acoustical Society of America*, vol. 98, no. 6, pp. 3302–3308, 1995.

[43] D. Murphy, A. Kelloniemi, J. Mullen, and S. Shelley, "Acoustic modelling using the digital waveguide mesh," *IEEE Signal Processing Magazine*, vol. 24, no. 2, pp. 55–66, 2007.

[44] P. Morse and U. Ingard, *Theoretical Acoustics*. Princeton, New Jersey: Princeton University Press, 1968.