TIMPANI DRUM SYNTHESIS IN 3D ON GPGPUS

Stefan Bilbao, [*]
Acoustics Group
University of Edinburgh
sbilbao@staffmail.ed.ac.uk

Craig Webb,
Acoustics Group/Edinburgh Parallel Computing Centre,
University of Edinburgh
C.J.Webb-2@sms.ed.ac.uk>

ABSTRACT

Physical modeling sound synthesis for systems in 3D is a computationally intensive undertaking; the number of degrees of freedom is very large, even for systems and spaces of modest physical dimensions. The recent emergence into the mainstream of highly parallel multicore hardware, such as general purpose graphical processing units (GPGPUs) has opened an avenue of approach to synthesis for such systems in a reasonable amount of time, without severe model simplification. In this context, new programming and algorithm design considerations appear, especially the ease with which a given algorithm may be parallelized. To this end finite difference time domain methods operating over regular grids are explored, with regard to an interesting and non-trivial test problem, that of the timpani drum. The timpani is chosen here because its sounding mechanism relies on the coupling between a 2D resonator and a 3D acoustic space (an internal cavity). It is also of large physical dimensions, and thus simulation is of high computational cost. A timpani model is presented, followed by a brief presentation of finite difference time domain methods, followed by a discussion of parallelization on GPGPU, and simulation results.

1. INTRODUCTION

A major recent research direction has been the large scale 3D simulation of acoustic spaces, for purposes of artificial reverberation or concert hall prototyping, using grid based methods such as the digital waveguide mesh [1] or finite difference time domain methods (FDTD) [2, 3]. The great advantage of such brute force techniques, compared with other room modelling methods, such as image source [4] or ray tracing [5] methods is that the acoustic field is modelled in its entirety, and without simplifying assumptions. The disadvantage, of course, is computational cost—for example, the simulation of a cubic metre volume, at 44.1 kHz, requires on the order of 10^{11} arithmetic operations per second of output. The simulation of large acoustic spaces at audio rates in a reasonable amount of time remains a challenging problem.

Smaller scale problems in 3D, however, are becoming tractable, especially on new highly parallel computer architectures. One application is 3D physical modeling synthesis, where a musical instrument is housed, or embedded, within a small 3D enclosure, intended not to emulate the behaviour of a room, but to allow complete modeling of the interaction between the instrument and the acoustic field, with as few modeling assumptions as possible. The timpani drum, or kettledrum is a good match to this problem, as the timbre of the instrument is crticially dependent on an enclosed volume, as well as the usual acoustic radiation phenomena, both of which are fully captured in a 3D model. The model used here is similar to that used by Rhaouti and Chaigne [6] in musical acoustics investigations, with some additional features. The computational algorithm presented here is geared towards a synthesis implementation in a highly parallel architecture, and is intended as a non-trivial feasibility test for the use of general purpose grapnical processing units (GPGPUs) in physical modeling sound synthesis.

A model of a single timpani drum is described in Section 2, including a description of the dynamics of the acoustic field, and its termination (by the drum cavity and an absorbing layer), and the membrane, including nonlinear effects, as well as the coupling between the two. Section 3 continues with a description of an FDTD algorithm operating over regular grids, as well as interpolation necessary to couple the two key subsystems. Implementation in CUDA on an Nvidia Tesla general purpose graphical processing unit is described in Section 4, followed by simulation results in Section 5.

2. MODEL SYSTEM

The system describing the timapani drum used here is similar to that used previously [6], and also in the case of the snare drum [7]. Using such a full model allows the direct simulation of acoustic radiation and cavity modes, as well as full spatialization; such features are difficult to capture using a 2D model [8], which, however, is far less computationally demanding in simulation.

The model is defined over a 3D volume $\mathcal{V}$, here taken to be a rectangular parallelepiped. It possesses an external boundary, $\partial\mathcal{V}_E$, consisting, in this case, of the six faces of the parallelpiped, and an inner boundary, consisting of the rigid shell boundary of the timpani drum, $\partial\mathcal{V}_S$, and the membrane, $\partial\mathcal{V}_M$. The acoustic field is defined on both sides of these latter two boundaries (i.e., inside and outside the cavity). See Figure 1.

2.1. Membrane

The primary vibrating component in the timpani, and all drums, is the drumhead itself. It is assumed to be a flexible membrane, defined over a two dimensional region $\partial\mathcal{V}_M$ (here a circle of radius R, defined, for simplicity at coordinate $z = z_M$ relative to the three dimensional acoustic space), with displacement constrained to be perpendicular to the plane in which it lies. The dynamics of such a membrane are described by the following equation:

$$\begin{aligned}\frac{\partial^2 w}{\partial t^2} \quad &= \quad c_M^2\left(1+g\right)\nabla_{2D}^2 w - \kappa^2\nabla_{2D}^2\nabla_{2D}^2 w + c_M^2\alpha\nabla_{2D}^2\frac{\partial w}{\partial t} \\ &+\frac{1}{\rho_M}\left(f_+ + f_-\right) + \frac{1}{\rho_M}\delta(x-x_0, y-y_0)f_{exc} \qquad (1)\end{aligned}$$

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

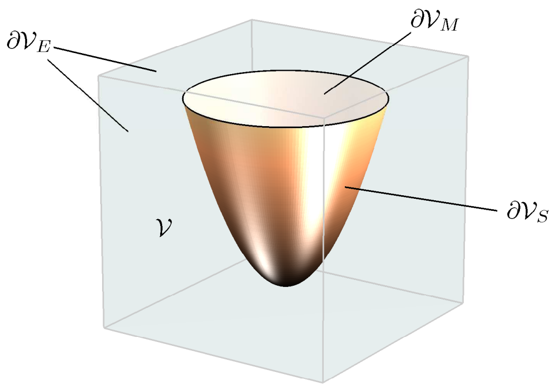

Figure 1: *Timpani drum geometry.*

where $w(x, y, t)$ is transverse displacement as a function of time t and coordinates x and y, and the 2D Laplacian operator is defined as

$$\nabla^2_{2D} = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \tag{2}$$

Here, ρ_M is the surface density of the membrane, in kg/m^2, and the wave speed c_M and stiffness parameter for the membrane are defined as

$$c_M = \sqrt{T_0/\rho_M} \qquad \kappa = \sqrt{EH^3/12\rho_M(1-\nu^2)} \tag{3}$$

where T_0 is membrane tension/unit length, in kg/s^2, H is membrane tickness, in m, E is Young's modulus, in kg /s^{2}m, and ν, dimensionless, is Poisson's ratio. α is a parameter determining viscothermal damping effects in the membrane.

The factor g represents an averaged nonlinearity in the membrane, and is defined as

$$g = \frac{EH}{2T_0(1-\nu^2)} \iint_{\mathcal{D}} |\nabla_{2D} w|^2 d\sigma \tag{4}$$

and results from a simplification (due to Berger [9]) of the von Karman system [10]. It may be viewed in the same light as the similar term appearing in the Kirchhoff Carrier equation [11] describing nonlinear vibration of a string, and gives rise, again, to pitch glide phenomena under high striking amplitudes, and which may dominate, as in the case of the string, at very low tension [12]. Such a model has been used previously in studies of the snare drum [8].

The terms in f_+ and f_- represent forcing due to coupling with the acoustic space, above and below the membrane, respectively. These will be related explcitly to the acoustic field in Section 2.5. The term in f_{exc} represents a pointwise excitation acting at location (x_0, y_0) on the membrane. See Section 2.6.

The membrane is assumed rigidly terminated (clamped), i.e.,

$$w = \mathbf{n}_{M,ext} \cdot \nabla_{2D} w = 0 \tag{5}$$

where $\mathbf{n}_{M,ext}$ represents the 2D normal vector to the outer boundary of the membrane, and ∇_{2D} is the 2D gradient operation.

2.2. Acoustic Field

Variation in the acoustic field is assumed here to satisfy the 3D wave equation:

$$\frac{\partial^2 \Psi}{\partial t^2} = c^2 \nabla^2_{3D} \Psi \tag{6}$$

where here, $\Psi(x, y, z, t)$ is a velocity potential [13] depending on time t and spatial coordinates x, y and z, and defined over the region $\mathcal{V}$. c is the wave speed in air, set here to 340 m/s, and ∇^2_{3D} is the 3D Laplacian operator, defined as

$$\nabla^2_{3D} = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \tag{7}$$

Ψ may be related to the more familiar quantities pressure p and particle velocity v by

$$p = \rho \frac{\partial \Psi}{\partial t} \qquad v = -\nabla_{3D} \Psi \tag{8}$$

where ρ is the density of air (taken here to be 1.2 kg/m^3), and ∇_{3D} is the 3D gradient operation.

For problems defined over small regions, as is the case here, the 3D wave equation is a good approximation to wave propagation in air. In larger regions, however, where sound propagates over a reasonable distance, it is important to include the effects of viscosity, which may be modelled using an extra term in (6) [13, 14].

Boundary conditions, for the external, shell and membrane boundaries will be given in the next sections.

2.3. Drum Cavity

The timpani drum shell is assumed rigid. The exact geometry is immaterial from a computational point of view, but it is assumed here to be paraboloidal, defined by the surface

$$z = z_M - H_0 + H_0 \left(x^2 + y^2\right)/R^2 \qquad z_M - H_0 \leq z \leq z_M \tag{9}$$

In other words, it is a section of a paraboloid, of height H_0 m, and operning up to a maximum radius of R m at vertical location $z = z_M$.

As the cavity is assumed to be rigid, a Neumann (zero normal velocity) condition is used, i.e.,

$$\mathbf{n}_S \cdot \nabla_{3D} \Psi = 0 \qquad \text{over} \qquad \partial\mathcal{V}_S \tag{10}$$

where $\mathbf{n}_S$ is the normal to the shell surface; the condition above holds on both faces of the shell (i.e., both inside and outside the cavity).

2.4. Absorbing Boundary Conditions

As the enclosure $\mathcal{V}$ is intended to be transparent to outward propagating waves, an absorbing boundary condition is necessary on the boundary $\mathcal{V}_E$. The usual choice is the well-researched perfectly matched layer [15], translated from the electromagnetic context to acoustics. In the interest of keeping programming complexity as simple as possible—a key concern in the development of methods on specialized hardware—a simpler choice of a boundary condition of Engquist Majda type [16] has been employed here. If $\mathbf{n}_E$ signifies an outward normal to a planar boundary, then the conditions may be written elegantly as

$$\left(\frac{\partial}{\partial t} + c\mathbf{n}_\mathbf{E} \cdot \nabla_{3D}\right)^q \Psi = 0 \tag{11}$$

in terms of the order q. Such conditions may always be rewritten, through substitution of (6), in terms of spatial derivatives tangential to the boundary, and a normal derivative of maximal order 1.

Such conditions inhibit reflection over an increasingly large range of frequencies and angles relative to the normal as q increases, but are not perfectly matched, for any finite order q (though in pratice, under discretization, neither are perfectly matched layers). As a result, spurious reflection by this boundary condition back into the problem domain is confined to high frequency waves which are tangential to, and thus remain trapped near the boundary itself. Though no perceptual testing has been done, it is clear that provided output is not drawn directly adjacent to the boundary itself, the conditions are perceptually transparent, at audio sample rates, even with an order as low as $q = 2$.

2.5. Coupling Conditions

At the interface $\partial\mathcal{V}_M$ between the membrane and the acoustic space, coupling conditions are necessary, and determine both the boundary condition for the acoustic field, as well as the excitation terms in (1).

If the velocity potentials on the upper and lower faces of the membrane are referred to as Ψ_+ and Ψ_- respectively, the conditions may be written as

$$f_- = \rho\frac{\partial\Psi_-}{\partial t} \qquad f_+ = -\rho\frac{\partial\Psi_+}{\partial t} \tag{12}$$

and

$$\frac{\partial w}{\partial t} = -\mathbf{n}_{M,+}\cdot\nabla\Psi_+ = \mathbf{n}_{M,-}\cdot\nabla\Psi_- \tag{13}$$

where $\mathbf{n}_{M,+} = -\mathbf{n}_{M,-}$ are the 3D outward unit normals to the membrane in the positive and negative directions. These pairs of conditions indicate continuity of pressure and velocity, respectively across the membrane interface.

2.6. Excitation and Output

Ideally, excitation should be modelled using a model of the mallet itself, subject to forcing by the player. Most models in the literature treat the mallet interaction in terms of an initial mallet velocity, and a nonlinear interaction between the mallet and the membrane. This approach well models a single strike—however, modeling of a full gesture (such as a drum roll, where there is a constant excitation force from the player) is far more complex. For this reason, and from the observation that (a) the interaction time between mallet and membrane is typically very short, it is sufficient to employ an external (non-interactive) forcing function $f_{exc} = f_{exc}(t)$ of a specified form. See [17] for more on this hypothesis. A good choice is of a raised cosine, for which amplitude and duration may be adjusted to striking strength. The spatial region of interaction is small, relative to audible wavenumbers in the membrane, and has been modelled here as a simple spatial Dirac delta function at the striking location.

3. FDTD ALGORITHM

In this section, finite difference time domain methods on regular grids are developed for the timpani model, as described in the previous section. FDTD methods are, of course, not the only approach to simulation—see, e.g., [6] for methods on irregular grids, and [8], where a similar system (the snare drum) is approached using modal techniques.

Schemes on regular grids, despite some awkwardness (namely in treating curved boundaries, and in interfacing between grids of different densities, for the membrane and acoustic space) are an excellent match to parallel architectures, as the use of an identical update at a large number of locations leads to the association with simple computational kernels (threads). Such is the case for spatially invariant systems such as a uniform membrane and acoustic field in Cartesian coordinates.

3.1. Membrane Scheme

The grid function $w_{l,m}^n$ is intended as an approximation to the membrane displacement $w(x, y, t)$, at coordinates $x = lh_M$, $y = mh_M$, and at time $t = nk$, for integer l, m and n, and where h_M is the grid spacing, and k is the time step (generally defined a priori, as $k = 1/F_s$, where F_s is the sample rate).

A finite difference scheme for the membrane, as defined in (1) may be written as

$$\delta_{tt}w = c_M^2(1+g)\delta_{2D}^2 w - \kappa^2\delta_{2D}^2\delta_{2D}^2 w + c_m^2\alpha\delta_{2D}^2\delta_{t-}w \tag{14}$$
$$+\frac{1}{\rho_M}(f_+ + f_-) + \frac{1}{\rho_M}\delta(x-x_0, y-y_0)f_{exc} \tag{15}$$

where w now stands for the grid function $w_{l,m}^n$, and where the various difference operators are defined as

$$\delta_{tt}w_{l,m}^n = \frac{1}{k^2}\left(w_{l,m}^{n+1} - 2w_{l,m}^n + w_{l,m}^{n-1}\right) \tag{16a}$$
$$\delta_{t-}w_{l,m}^n = \frac{1}{k}\left(w_{l,m}^n - w_{l,m}^{n-1}\right) \tag{16b}$$
$$\delta_{2D}^2 w_{l,m}^n = \frac{1}{h_M^2}\Big(w_{l+1,m}^n + w_{l-1,m}^n + w_{l,m+1}^n + w_{l,m-1}^n - 4w_{l,m}^n\Big) \tag{16c}$$
$$\mu_{t\cdot}w_{l,m}^n = \frac{1}{2}\left(w_{l,m}^{n+1} + w_{l,m}^{n-1}\right) \tag{16d}$$
$$\delta_{t\cdot}w_{l,m}^n = \frac{1}{2k}\left(w_{l,m}^{n+1} - w_{l,m}^{n-1}\right) \tag{16e}$$

and the other operators in (14), namely $\delta_{2D}^2\delta_{2D}^2$ and $\delta_{2D}^2\delta_{t-}$ may be formed through composition of the operators defined above. The clamped boundary conditions (5) may be simply approximated by a staircase approximation, assuming that values of the grid function which lie outside the membrane boundary are permanently set to zero. See Section 5.1 for the consequences of this (crude!) numerical boundary condition.

The scalar nonlinear term $g = g^n$ requires a special treatment, in line with that of averaged nonlinearities which appear in the case of the string [18]. An approximation consistent with (4) is the following:

$$g = \frac{EH}{2T_0(1-\nu^2)}\sum_{\partial\mathcal{V}_M^{(d)}} h_M^2\nabla_{2D}^{(d)}w\cdot\nabla_{2D}^{(d)}\mu_{t\cdot}w \tag{17}$$

where $\mu_{t\cdot}$ is the averaging operator given in (16d), and the difference approximation $\nabla_{2D}^{(d)}$ to the 2D gradient is defined as

$$\nabla_{2D}^{(d)}f_{i,j} = \frac{1}{h_M}[f_{i,j} - f_{i-1,j}, f_{i,j} - f_{i,j-1}] \tag{18}$$

and summed over the region $\partial\mathcal{V}_M^{(d)}$, consisting of all grid locations l, m such that $h_M l$, $h_M m$, $h_M(l-1)$ and $h_M(m-1)$ lie within a circle of radius R.

Though apparently implicit (i.e., when applied at time step n, the averaging operation $\mu_{t\cdot}$ introduces values of the as yet unknown values of the grid function at time step $n+1$), due to the scalar nature of the nonlinearity, this expression may be written explicitly in terms of known values of w at time step n. See [17] for more details.

The coupling terms f_+ and f_- are now 2D grid functions, i.e., $f_+ = f^n_{+,l,m}$ $f_- = f^n_{-,l,m}$ and and the external excitation f_{exc} is a specified time series $f_{exc} = f^n_{exc}$. The grid function $\phi = \phi_{l,m}$ is an approximation to a 2D Dirac delta function—if coincident with a grid location $l = l_0, m = m_0$, it takes on the value $1/h_M^2$ at this location, and is zero otherwise. If not, a bilinear, or higher order interpolant may be used.

3.2. Scheme over Acoustic Field

Over the acoustic field, a simple seven point scheme is used:

$$\delta_{tt}\Psi = c^2\delta^2_{3D}\Psi \tag{19}$$

for a grid function $\Psi^n_{l,m,p}$ approximating the acoustic field $\Psi(x,y,z,t)$ at coordinates $x = lh, y = mh, z = ph$ and at time $t = nk$, where h is the grid spacing. Here the operator δ_{tt} is as defined in the 2D case in (16), and the 3D Laplacian difference operator as

$$\delta^2_{3D}\Psi^n_{l,m,p} = \sum_{\substack{\alpha,\beta,\gamma \\ |\alpha|+|\beta|+|\gamma|=1}} \Psi^n_{l+\alpha,m+\beta,p+\gamma} - 6\Psi^n_{l,m,p} \tag{20}$$

At the interface with the drum cavity, a staircase approximation is again is used. At a grid location l_b, m_b, p_b, for example (either inside or outside the cavity), for which one or more adjacent grid locations lie on the other side of the cavity boundary, the Laplacian is specialized to

$$\begin{aligned}\delta^2_{3D}\Psi^n_{l_b,m_b,p_b} = & \sum_{\substack{\alpha,\beta,\gamma \\ |\alpha|+|\beta|+|\gamma|=1}} \left(\Psi^n_{l_b+\alpha,m_b+\beta,p_b+\gamma}\right)^{(int)} \\ & -K_{l_b,m_b,p_b}\Psi^n_{l_b,m_b,p_b}\end{aligned} \tag{21}$$

where the superscript $^{(int)}$ restricts the sum to values which lie on the same side of the boundary as the grid point l_b, m_b, p_b, and where the integer K_{l_b,m_b,p_b} is the number of nearest neighbours lying on the same side of the boundary.

For the membrane coupling conditions, it is easiest (though by no means necessary) to place the membrane height z_M at a location directly between two vertical sets of grid points, with index p_+ and p_-, and thus at spatial locations $z_M + h/2$ and $z_M - h/2$, respectively. The coupling conditions (12) become:

$$f_- = \rho\delta_{t\cdot}\mathcal{I}_{3D\to 2D}\Psi_{-,p_-} \qquad f_+ = -\rho\delta_{t\cdot}\mathcal{I}_{3D\to 2D}\Psi_{+,p_+} \tag{22}$$

where $\Psi_{-,p_-} = \Psi^n_{l,m,p_-}$ and $\Psi_{+,p_+} = \Psi^n_{l,m,p_+}$, and where $\delta_{t\cdot}$ is a centered time difference operator, as given in (16e). The operation $\mathcal{I}_{3D\to 2D}$ interpolates grid values for the acoustic field to locations on the grid for the membrane.

Coupling conditions (13) become, under simple difference operations,

$$\begin{aligned}\mathcal{I}_{2D\to 3D}\delta_{t\cdot}w^n &= -\frac{1}{h}\left(\Psi^*_{-,p_-+1} - \Psi_{-,p_-}\right) \\ &= -\frac{1}{h}\left(\Psi_{+,p_+} - \Psi^*_{+,p_+-1}\right)\end{aligned} \tag{23}$$

where Ψ^*_{-,p_-+1} and Ψ^*_{+,p_+-1} indicate virtual values of the acoustic field, and where $\mathcal{I}_{2D\to 3D}$ is an interpolant from membrane displacement values on the 2D grid to grid locations in 3D.

The conditions (22) and (23), when coupled with the membrane update (14) and (19), lead to a full update of both fields; furthermore, through inbuilt numerical energy conservation (see Section 3.3).

Numerical equivalents of the absorbing boundary conditions have been presented by one of the authors recently, in the case of the snare drum [7].

3.3. Energy Conservation and Stability Conditions

In the case of a nonlinear coupled system such as this, numerical stability is best approached using energy techniques [19]. As this topic has been covered at length by one of the authors (particularly in [17]), a lengthy review is unnecessary here.

The idea is that under lossless conditions (here, when $\alpha = 0$, and when absorbing boundary conditions are disabled), the total energy of the system must remain constant under transient conditions. If such a property of an invariant energy can be built into the numerical simulation method, and, furthermore, the numerical energy is positive, then the algorithm is numerically stable; this includes the effect of the nonlinearity in the membrane. See Section 5.3 for an illistration of numerical energy conservation to machine accuracy.

Positivity of numerical energy is ensured [17] under the stability conditions

$$h_M^2 \geq c_M^2k^2 + 2c_M^2\alpha k + \sqrt{\left(c_M^2k^2 + 2c_M^2\alpha k\right)^2 + 16\kappa^2k^2} \tag{24}$$

and

$$h \geq \sqrt{3}ck \tag{25}$$

if the two interpolants $\mathcal{I}_{2D\to 3D}$, and $\mathcal{I}_{3D\to 2D}$ are chosen such that when written in matrix form, they are transposes of one another, scaled by h_M/h. These stability conditions are identical to those which would be otained, under von Neumann analysis, for the linear membrane system and acoustic space in isolation—now, however, they hold in the case of a coupled nonlinear system.

4. PARALLEL IMPLEMENTATION USING CUDA

Implementation the timpani model described above inevitably relies on linear algebra constructs using both dense and sparse matrix operations. Initial prototyping was performed using MATLAB, resulting in some five hundred lines of matrix based code, the majority of which is pre runtime loop setup (primarily in generating matrix equivalents of the various difference operators). Benchmarking on an Intel Core i7 "Sandy Bridge" processor gave a computation time of 28.0 minutes for one second of simulation at 44.1kHz, for a typical drum enclosed within a cubic metre volume. This is clearly not ideal for the user in learning how to play an instrument.

One of the benefits of using FDTD schemes is their high level of data independence at each time step. Such schemes are well suited to parallel processing using GPUs, which can dramatically reduce computation times even when using double precision calculation [20]. This section details the transition from MATLAB to the use of Nvidia's CUDA architecture.

4.1. Algorithm Structure

A brief overview of the algorithm is useful in understanding the computational requirements.

Preprocessing consists of matrix and grid space setup, and creating a list of grid points adjacent to the cavity boundaries. This takes a matter of seconds and is therefore minimal compared to computation time in the main kernel. The run-time loop performs the following operations at each time step:

1. Calculate the 3D Laplacian taking into account cavity and absorbing boundaries.
2. Compute the membrane displacement, with iterative linear system solutions and interpolation.
3. Adjust acoustic field for interpolated effect of membrane interaction.
4. Read output from selected locations in the 3D grid.

Due to the implicit character of the coupling conditions (**??**), a linear system solution is required to update the membrane displacement, and an iterative method (provably convergent) using five to ten iterations gives acceptable accuracy for correctness testing. Table 1 shows the sizes of the grids used by the two components of the system.

System	Dimensions	Total grid points
2D membrane	127 x 127	16,129
3D volume	75 x 75 x 75	421,875

Table 1: System grid sizes at 44.1kHz

The MATLAB implementation consists almost entirely of large sparse matrix operations and other grid updates performed in a vectorized manner. This is optimal in terms of runtime, and also efficient in terms of code complexity. However, this does present some challenges when porting to C and CUDA.

4.2. Translation to CUDA Code

For FDTD schemes, the key to maximising performance on a GPU is *reducing* and *coalescing* access to global memory (coalescing is achieved when consecutive threads in a group access contiguous elements of global memory). GPUs are very good at performing floating point operations, which can lead to a bottleneck when transferring the data on and off the registers. This is a limiting factor for FDTD, where the compute/memory access ratios are very low. Given this limitation, the structure of the MATLAB code, with its reliance on large but highly sparse matrices of repeated coefficients, is certainly not optimal for CUDA.

Refactoring of the code to "unroll" very large sparse matrix operations into explicit loop-based updates using scalar coefficients constituted the first stage of the port to CUDA. This was performed for the 3D volume calculations. The 2D membrane calculations due to interpolation, are much harder to rewrite in a loop-based form, and were thus left in sparse matrix form. The result is a hybrid system of explicitly threaded and matrix form computations, the latter requiring the use of objects handling sparse matrices in both C and CUDA.

For the initial setup of sparse matrices in C code, elements of the CSparse library were used [21]. This is a CSC format library that uses efficient, but single threaded functions that replicate those used in MATLAB. For the membrane update calculations in the kernel, Nvidia's CUSPARSE v2 library provides the necessary sparse matrix by dense vector multiplication [22]. This requires the CSR format, so the matrices created in CSparse were transposed and the row and column arrays reversed.

Whilst the libraries provide the necessary tools for porting the MATLAB, there are still issues that need to be addressed to obtain GPU performance.

4.3. GPU efficiency

Operations that are computationally expensive in serial C code can be straightforward to run in CUDA. For instance, calculating the 3D Laplacian (Step 1.) in C code requires looping over the large volume size, and computing an update for each point based on its six nearest neighbours. This is naturally an expensive operation. However, on the GPU it is very simple to code and runs extremely fast. Using the latest FERMI architecture cards, there is no need to use shared memory for small scale stencils as the caching system performs the same function [23]. CUDA v4 allows thread grids which are three dimensional, and so there are no issues in mapping the data to the grid.

In contrast, operations that are trivial in serial C can become slightly tricky to perform in CUDA. Consider `y = a'*a`, for a dense column vector multiplied by its transpose to give a scalar value result. This is used several times in the membrane calculation, including in the linear system solve. In C code, this is simply a case of looping over the vector, multiplying each value by itself and summing it into a variable holding the cumulative total. As the 2D size is only 16 thousand points, this is an extremely fast calculation.

Moving to CUDA, parallelizing this operation is not readily apparent due to the cumulative sum. This first thing to note is that this needs to be preformed on the GPU, as transferring data between host CPU and the device has the slowest transfer rates. So, an initial "naive" approach would be to get a single thread to perform the computation as per the serial C code.

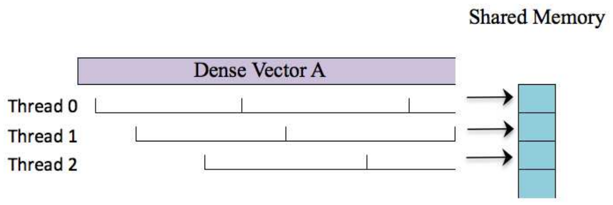

Figure 2: *Vector multiplied by its transpose using shared memory.*

This turns out to be extremely slow, even though the thread only loops over a relatively small number of values, as no memory coalescing occurs. To improve on this, one can use a reduction algorithm technique that utilises shared memory (fig.2). This is a two stage approach. Firstly, issue $N = 1,024$ threads in a single thread block. Each thread then "strides" over the vector elements in a loop, shifting by N elements at each pass, multiplying the element by itself and adding the result to the cumulative sum in register. Each thread then stores its partial sum into a shared memory array of size N. The second stage is to get a single thread to then loop over this shared memory array and sum the final result. This approach achieves memory coalescing for the data read from global memory, and leaves a single thread looping over a small amount of shared memory which is fast.

There are other stages in the kernel where memory coalescing issues occur, and for instance in adjusting the Laplacian for the cavity boundaries. These boundary points are stored in a list of, in the case of a typical drum modelled at 44.1 kHz, around 10 thousand rows and ten columns. This list is iterated through at stage 1. of the kernel, picking-up values in the 3D volume. Achieving coalescing on the reads from the volume is hard, as they refer to a 3D parabolic shape. However, coalescing the iteration through the list is straightforward, again using a "strided" access approach and ensuring that the list is linearly decomposed to enable this.

5. SIMULATION RESULTS

5.1. Accuracy of Staircase Approximation

As a preliminary check, it is useful to examine the use of the staircase approximation in the case of the membrane. A plot of modal frequencies, for an ideal membrane (i.e., without losses, stiffness or nonlinearity) appears in Figure 3, accompanied by an output spectrum for a membrane simulation with a staircase approximation to the boundary. As expected, low frequency modes are approximated nearly exactly, with increasing error, due to numerical dispersion, for increasing frequency.

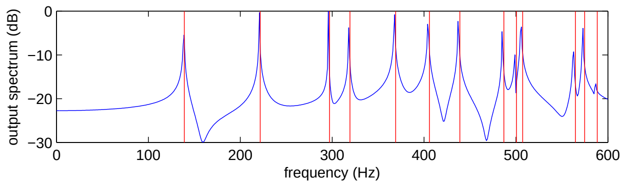

Figure 3: *In red: Exact modal frequencies for an isolated circular membrane, with wave speed $c_M = 112.65$ m/s, under linear, lossless and non-stiff conditions, and fixed boundary conditions. In blue: Output spectrum calculated through FDTD on a Cartesian grid, using a staircase approximation at the boundary.*

5.2. Pitch Glides

Pitch glides, though a minor effect in timpani, can be reproduced using this model—an exaggerated pitch glide is illustrated in Figure 4.

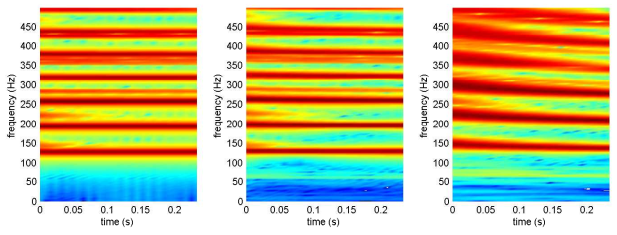

Figure 4: *Spectrogram for sound output from the timapni model, under low, medium, and high striking amplitudes; a pitch glide phenomenon at high striking amplitude is apparent.*

5.3. Numerical Energy Conservation

As an illustration, variation in numerical energy for this system, subjected to an impulsive excitation, is shown in Figure 5; energy is constant to machine accuracy, as is clearly visible as bit quantization in the energy. Under the stability conditions (24) and (25), its constancy serves as a stability condition for the scheme as a whole.

Though there is no need to calculate energy explicitly in a polished simulation, such an energy measure is a useful feature in debugging, and serves as a form of checksum—virtually any error in the implementation will be reflected by energy variation.

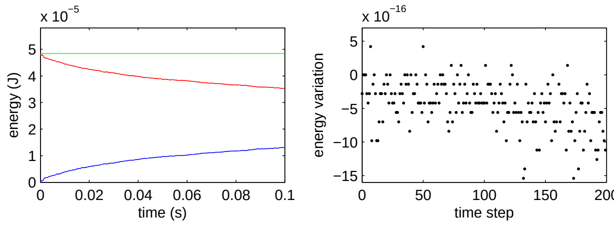

Figure 5: *Left, numerical energy of membrane (red), acoustic field (blue) and total (green) as a function of time. Right, variation in total numerical energy, normalized by total energy.*

5.4. Performance analysis

The CUDA code is some three times longer than in MATLAB, at 1,500 lines (excluding the libraries). The Nvidia Tesla C2050 card was used for performance testing, using double precision floating point calculations. Running at 44.1kHz, this gave a computation time of 82 seconds for a typical timpani drum set within a cubic metre volume, a $\times$ 20.5 speedup over the original MATLAB code. Computation time for a three second simulation is reduced to 4 minutes, compared to nearly an hour and a half in MATLAB. Timings for the individual parts of the computation are given in Table 2.

Kernel	Time (% of total)
Calculate Laplacian over 3D volume	8.50 %
Membrane : update	17.74 %
Membrane : interpolation	5.91 %
Membrane : linear system solution	48.76 %
Update volume interior	9.58 %
Update volume boundaries	7.31 %
Membrane coupling	1.95 %
Read output	0.25 %

Table 2: Relative kernel times for one time step at 44.1kHz

Despite the relatively small size of the 2D membrane grids, their update takes 70% of the total kernel time. The iterative linear system solution is clearly the stalling point, taking nearly half of the entire time. Each iteration requires two large scale CUSPARSE kernel calls, two simple kernel updates, and a reduction algorithm kernel as detailed above. This leaves little room for optimization in its current form. However, further refactoring of the code to a loop based implementation the membrane sparse matrix multiplications would most likely lead to further reductions in the computation time.

5.5. Full 3D Simulation Results and Sound Examples

For the sake of illustration, spashots of a cross section of the acoustic field are shown in Figure 6. Sound examples produced using this model are available at

http://www2.ph.ed.ac.uk/~s0956654/Site/Instruments.html

6. CONCLUDING REMARKS

The timpani drum model described here is mainly linear, and gives good quality results under low striking amplitudes. It is clear, however, from audio output, that the model does not adequately capture the salient perceptual qualities of the timpani at high striking amplitudes. This suggests a further, as yet unmodelled nonlinearity, and a good candidate for further investigation is the membrane itself—an averaged tension modulation term is an approximation to a fuller nonlinear model, which, in addition to producing pitch glides, will also serve to generate a cascade of energy towards the high frequency range. Under extreme striking amplitudes, the possibility that nonlinear effects in the air itself (i.e. shocks) are generated within the cavity should be carefully considered. Other more minor features which could be added are variable tension, both spatially and temporally, in the membrane itself, and the ability of the shell to vibrate. From a numerical design point of view, the approximations to curved boundaries (i.e. staircase) used here are rather crude, but will only lead to artifacts at wavenumbers on the order of the grid resolution—in particular, the lower modal frequencies (especially important in the case of the membrane, and the cavity) are very well simulated.

The development of a fully 3D physical modeling synthesis environment requires two further steps: one is to extend the 3D computation region to that of an entire acoustic space, which would require a more detailed treatment of the boundary ∂V_E, along the lines of room acoustics simulation [1], and the embedding of multiple such 3D instruments in the same space (allowing more subtle effects such as sympathetic vibration). Both topics are the subject of ongoing research at the University of Edinburgh.

From a computational point of view, the main point that is made in this article is that is becoming possible to simulate relative complex systems, with a minimum of corner-cutting hypotheses in a reasonable amount of time. Algorithm design for such systems is a delicate undertaking—in an audio setting, the main concerns are the usual ones: (a) maintaining numerical stability under the wide vartiety of instrument design and playing conditions specified by a user, and (b) maintaining fidelity, at the perceptual level, to the underlying physical model, in the face of effects such as numerical dispersion. In parallel harware, however, these issues are complemented by (c) new parallelization issues and their influence on computation time, which interacts with and informs the basic design issues (a) and (b). An interesting example of this is afforded by the case of coupled disparate grids, requiring interpolation. For problems defined over relatively small geometries, in the case of the present algorithm, this interpolation turns out to be a bottleneck, and can in fact be slower than the computation of the entire 3D acoustic field—which is opposite to the case of the algorithm running on conventional hardware. Two ways to counteract this might be to relax the energy-based requirement on stability, from (a) leading to a fully explicit update, or to choose coincident grids in the two spaces, leading to severe numerical dispersion, from (b). Neither seems acceptable, for an algorithm running at an audio rate.

As mentioned earlier, the timapni drum here has been taken to be a test case for the implementation of a non-trivial physical modeling synthesis algorithm in parallel hardware. The port from Matlab to CUDA was carried out, here, in a relatively direct fashion, without a serious attempt at optimization, and has resulted in a large speed-up—with further optimization, one could well expect a much greater acceleration. Yet, case-by-case optimization is a tedious undertaking, and efforts are being made at Edinburgh to develop parallel optimization tools which can be used for configurations of increasing generality beyond that of a single drum.

7. REFERENCES

[1] D. Murphy, A. Kelloniemi, J. Mullen, and S. Shelley, “Acoustic modelling using the digital waveguide mesh,” *IEEE Sig. Proc. Mag.*, vol. 24, no. 2, pp. 55–66, 2007.

[2] D. Botteldooren, “Finite-difference time-domain simulation of low-frequency room acoustic problems,” *J. Acoust. Soc. Am.*, vol. 98, no. 6, pp. 3302–3308, 1995.

[3] L. Savioja, D. Manocha, and M. Lin, “Use of GPUs in room acoustic modeling and auralization,” in *Proc. Int. Symposium on Room Acoustics*, Melbourne, Australia, Aug. 2010.

[4] J. Allen and D. Berkley, “Image method for efficiently simulating small-room acoustics,” *J. Acoust. Soc. Am.*, vol. 65, no. 4, pp. 943–950, 1979.

[5] T. Funkhouser et al., “A beam tracing method for interactive architectural acoustics,” *J. Acoust. Soc. Am.*, vol. 115, no. 2, pp. 739–756, 2004.

[6] L. Rhaouti, A. Chaigne, and P. Joly, “Time-domain modeling and numerical simulation of a kettledrum,” *J. Acoust. Soc. Am.*, vol. 105, no. 6, pp. 3545–3562, 1999.

[7] S. Bilbao, “Time domain simulation of the snare drum,” *J. Acoust. Soc. Am.*, vol. 131, no. 1, pp. 914–925, 2012.

[8] R. Marogna and F. Avanzini, “A block-based physical modeling approach to the soundsynthesis of drums,” Under review, IEEE Transactions on Audio Speech and Language Processing, 2010.

[9] H. Berger, “A new approach to the analysis of large deflections of plates,” *Journal of Applied Mathematics*, vol. 22, pp. 465–472, 1955.

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

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

[12] B. Bank, *Physics-based Sound Synthesis of String Instruments Including Geometric Nonlinearities*, Ph.D. thesis, Budapest Univeristy of Technology and Economics, 2006.

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

[14] C. Webb and S. Bilbao, “Computing room acoustics with cuda - 3d fdtd schemes with boundary losses and viscosity,” in *Proceedings of the IEEE International Conference*

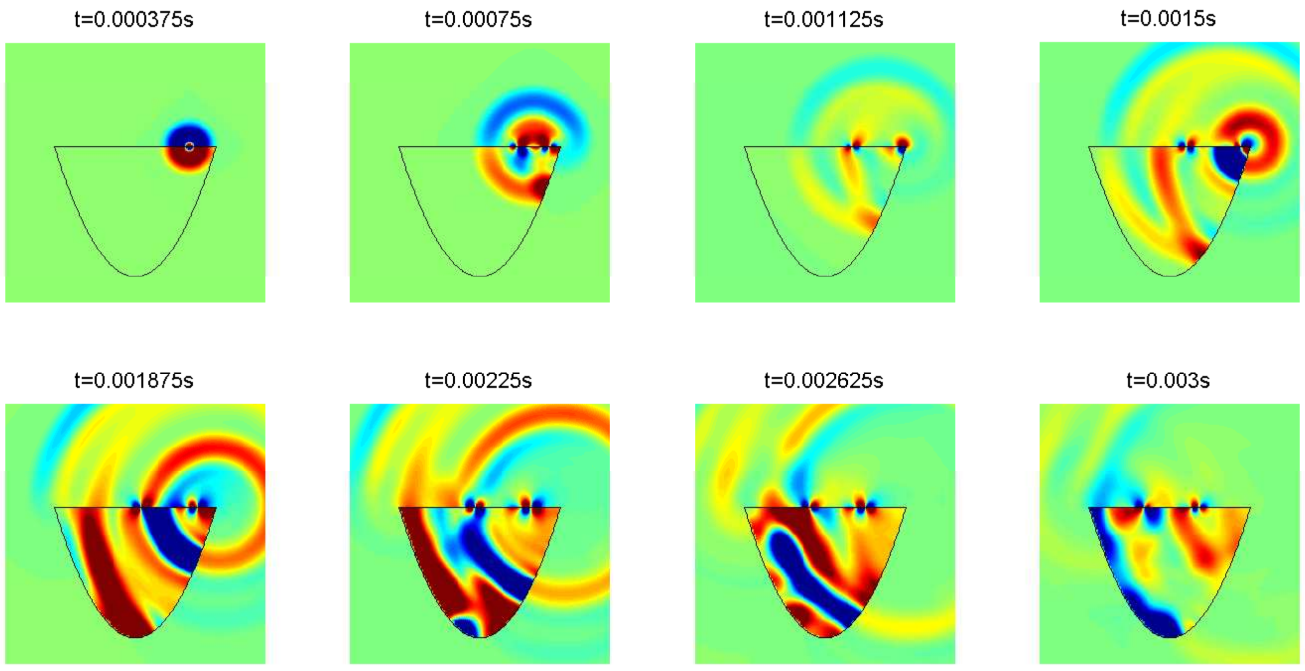

Figure 6: *Snapshots of the cross section of the pressure field for a timpani simulation, at times as indicated.*

on Acoustics, Speech, and Signal Processing, Prague, Czech Republic, 2011.

[15] J.-P. Berenger, "Three-dimensional perfectly matched layer for the absorption of electromagnetic waves," *J. Comp. Phys.*, vol. 127, no. 2, pp. 363–379, September 1996.

[16] B. Engquist and A. Majda, "Absorbing boundary conditions for the numerical evaluationof waves," *Mathematics of Computation*, vol. 31, no. 139, pp. 629–651, 1997.

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

[18] S. Bilbao and J. O. Smith III, "Energy-conserving finite difference schemes for nonlinear strings," *Acta Acustica united with Acustica*, vol. 91, no. 2, pp. 299–311, 2005.

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

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

[21] T. Davis, *Direct Methods for Sparse Linear Systems*, SIAM, 2006.

[22] Nvidia Corp, "CUDA Toolkit 4.1 CUSPARSE library user guide," Available : http://developer.nvidia.com/nvidia-gpu-computing-documentationcusparse, Jan 2012.

[23] Nvidia Corp, "Fermi compute architecture whitepaper," Available : http://www.nvidia.com/object/fermiarchitecture.html, Oct, 2010.