Modular Physical Modeling Synthesis Environments on GPU

Stefan Bilbao and Alberto Torin
Acoustics and Audio Group
University of Edinburgh
sbilbao@staffmail.ed.ac.uk
s1164558@sms.ed.ac.uk

Paul Graham and James Perry
Edinburgh Parallel Computing Centre
University of Edinburgh
paul@epcc.ed.ac.uk
j.perry@epcc.ed.ac.uk

Gordon Delap
Department of Music
NUI Maynooth
g.delap@googlemail.com

ABSTRACT

Physical modeling synthesis is a powerful means of access to a wide variety of synthetic sounds of an acoustic character—one longstanding design principle underlying such methods has been, and continues to be modularity, or the decomposition of a complex instrument into simpler building blocks. In this paper, various modular physical modeling design environments, based on the use of time stepping methods such as finite difference time domain methods are described, with an emphasis on the underlying computational behaviour of such methods, both in the run-time loop and in precomputation. As such methods are computationally intensive, additional emphasis is placed on issues surrounding parallelisation, and implementation in highly parallel hardware such as graphics processing units. This paper is paired with a recently completed multichannel piece, and the composer's perspective on working with such environments is also addressed.

1. INTRODUCTION

Physical modeling synthesis is a longstanding attempt to overcome the limitations of abstract and sampling-based synthesis methods; the aim is to allow the composer flexible access to a wide variety of sound material of an acoustic character. To this end, modularity is a useful design principle.

Modular physical modeling construction environments, whereby the user takes on the role not just of composer, but also instrument builder are by no means new. The first complete system, CORDIS, making use of lumped networks of masses and springs [1], has been under development since the late 1970s [2]. Another approach, based on the use of modal expansions for distributed objects such as strings, membranes, and plates, accompanied by a connection framework [3] is the basis for the MOSAIC and Modalys systems [4, 5] developed at IRCAM. Finally, digital waveguides [6], based on the use of traveling wave representations, and implemented efficiently as delay lines also allow for such modular constructions [7], particularly when complemented by the machinery of wave digital filters [8].

The methods mentioned above are all distinct (with some interesting points of contact) and all have strengths and weaknesses. The lumped network paradigm allows for very low level control (to the extent that the properties of individual masses are accessible), but the construction of distributed sound-producing objects, while possible, is more cumbersome. Modal methods can be extremely efficient for linear objects with a relatively low number of audible modal frequencies (such as, for instance, a marimba bar), but nonlinear behaviour becomes more difficult to model (though again, possible), and precomputation costs and storage for modal representations for nontrivial objects can be prohibitive. The waveguide framework yields extremely efficient implementations for certain elements such as an ideal string, or a uniform cylindrical or conical tube, but loses efficiency quickly when extended to more complex settings in higher dimensions or when nonlinearities are present.

Direct time-stepping methods, operating over grids, such as finite difference time domain methods [9], or finite volume methods [10], can be viewed as compromise allowing for generality and flexibility, especially in a modular environment, while sometimes sacrificing efficiency and, of course, introducing new problems all their own, such as the maintenance of numerical stability and audible artifacts due, e.g., to numerical dispersion, requiring some painstaking work at the design stage. For a fuller discussion of the distinctions between such time stepping methods and the other synthesis techniques listed above, from the perspectives of both computational cost and perceptual artifacts, see [11].

This paper describes ongoing work at the University of Edinburgh in large-scale, and ultimately 3D physical modeling synthesis under the umbrella of the NESS Project, particularly using direct PDE solvers, and parallelized in multicore and also on GPU. One aspect of such work is the construction of modular physical modeling synthesis systems, accompanied by work directly with composers. This paper is structured as follows: In Section 2, various components of a percussion-based modular synthesis system are described, including the surrounding 3D space as a single "component" of its own. Section 3 describes, in a general fashion, the application of grids and time stepping methods to such systems, resulting in recursions operating over a vectorized state, with a high-level description of the

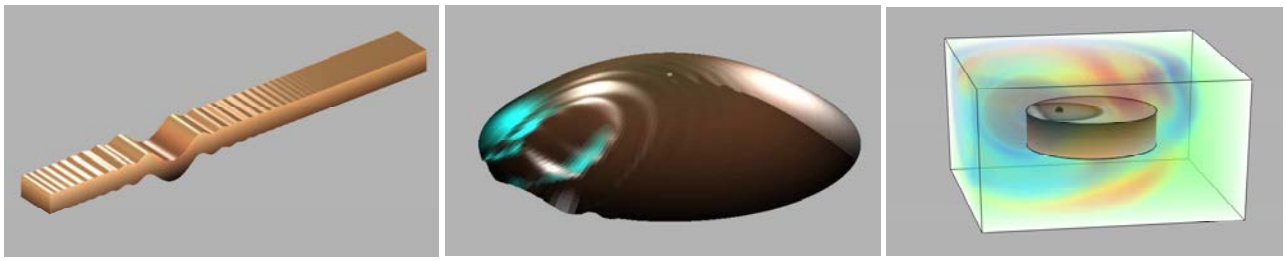

Figure 1. Left: vibrating 1D bar system. Center: Struck curved plate. Right: 3D acoustic field generated by striking a drum.

functioning of such algorithms in the run-time loop. Section 4 is a presentation of three particular large-scale modular synthesis environments; parallelization and the resulting port to multicore and GPU are described in Section 5, as well as a rudimentary interface in Section 6. Finally, the musician's point of view is considered in Section 7; after having performed extensive investigation of such environments, one of the authors (Delap) successfully created an original multichannel computer music piece, which forms the companion to this article.

2. COMPONENTS

In the framework described below, the basic sound-producing objects, or modules, are taken to be distributed—that is, occupying space in 1, 2, or 3 dimensions. Each such module is thus characterized by a relatively small number of material and geometric parameters. In this sense, it is similar to modal environments, but distinct from methods based on lumped mass/spring networks.

2.1 1D Objects: Strings and Bars

The most basic distributed objects of interest in a physical modeling synthesis framework are 1D objects such as strings and bars (wind instruments, based on acoustic tube models, are also well-modelled in 1D, and under development in the NESS project, but are not covered here). Such objects are assumed thin, so that the displacement may be written as a function of time t and a single spatial coordinate x. Strings are assumed to be without inherent stiffness, and support vibration due to tensioning; bars are assumed stiff, and untensioned; a stiff string incorporates both such effects. See Figure 1, at left.

There are many models of the vibration of such 1D systems [12], of differing levels of complexity, including effects of longitudinal vibration, vibration in different polarizations, non-negligible thickness, and also different degrees of nonlinearity, including averaged effects such as tension modulation, leading to pitch glides [13], and, in the most involved case, full nonlinear coupling among the transverse polarizations and longitudinal motion, leading to effects such as phantom partial generation in piano strings under high striking amplitudes [14].

2.2 2D Objects: Membranes, Plates and Shells

Thin 2D structures such as membranes and plates are analogous to the string and bar, respectively, and play a key role in most percussion instruments. Nonlinearity, if modelled, plays a much larger role than in the case of strings and bars, leading to dramatic effects such as crashes in gongs [15]. Curved structures (shells) can also be employed as models of instruments such as cymbals. See Figure 1, at center.

2.3 Embedding in 3D

For full 3D rendering, simulation of the acoustic field is necessary, and is the most computationally heavy (but also most parallelizable) operation. One major interest here is in embedding other components in 3D space, such as isolated plates, or cavity/membrane combinations as in the case of timpani [16] and snare drums [17], allowing for natural modeling of acoustic radiation, and also for spatialized audio output, drawn directly from the field. If an embedded instrument is to be modelled in isolation, a relatively small enclosing space can be employed, with absorbing boundary conditions used in order to reduce or eliminate spurious reflections. If an entire room surrounding the instrument is to be modelled, then realistic room impedance boundary conditions can be employed. See Figure 1, at right.

2.4 Connections

The view taken here of a connection is perhaps more general than in other physical modeling environments, in that it refers to any interchange of energy between individual components (including the acoustic field, in a 3D setting) where conservation of momentum is enforced, and where energy is conserved (or more generally dissipated). In particular, at the level of the resulting implementation, there is not a distinction between lumped connections, and those with a distributed character. What does have an impact on the implementation is nature of the connection is linear or nonlinear.

Linear connections include: a lumped linear mass-spring connection between two components, at specified locations, the pointwise connection of two distributed components (such as a string and plate, in the case of a soundboard), and also the interaction of a distributed object such as a plate with the acoustic field. Nonlinear connections include: the interaction of a lumped striking or bowing mechanism with a distributed object, ans also fully distributed interactions such as the collision of a membrane with a wire (in the case of a snare drum), or of a string with a fretboard or barrier. Linear and nonlinear connections are

handled differently in the resulting implementation—see Section 3.

3. TIME DISCRETIZATION, GRIDS, RECURSIONS AND THE RUN-TIME LOOP

In sound synthesis applications, it is convenient to discretize the entire system at a uniform sample rate f_s. When time stepping methods are employed, the choice of sample rate implies, for the simplest (explicitly computable) algorithms, a lower bound on the spatial grid size used to represent a particular component, and, in order to minimize perceptual artifacts resulting from numerical dispersion [9], it is best to choose the grid so as to satisfy this bound as close to equality as possible. In general, this implies that grids for different components will be distinct. See Figure 2 for a depiction of grids used in 1D, 2D and 3D—all Cartesian, here, as such regularity leads to greater ease in terms of parallelisation. It is important to note that the grid representation here is not to be interpreted as a network of lumped masses, as in the case of environments such as CORDIS—the variables calculated over such grids sometimes correspond to displacements of the medium, and sometimes to other variables (such as potential functions, bending moments, etc.).

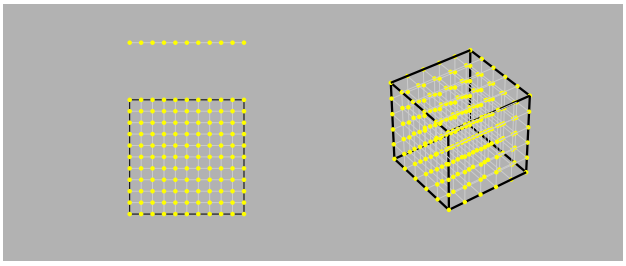

Figure 2. Regular grids used in the solution of systems in 1D, 2D and 3D.

Ultimately, regardless of the form of the modular construction, or the choice of grid, the state of the entire network may consolidated into a vector $\mathbf{u}$, and must be advanced in time at the audio sample rate. See, e.g., [18] for a more elaborate discussion on this topic. The resulting algorithm may be written in a generalized state-space form outlined in the pseudocode given below:

`Precomputation:`

- `From instrument definition:` $\mathbf{A}_l, \mathbf{B}_l, \mathbf{C}_l, \mathbf{B}_{nl}, \mathbf{C}_{nl}, \mathbf{M}_l, \mathbf{M}_{nl}, \mathbf{Q}$
- `From score:` $\mathbf{M}_e, \mathbf{f}_e^n$

`Initialization:` $\mathbf{u}_{last}$

`Run-time loop:`

`for n=1:final`

1. $\mathbf{A} = \mathbf{A}(\mathbf{u}_{last}), \mathbf{B} = \mathbf{B}(\mathbf{u}_{last})$
2. $\mathbf{A}\mathbf{u} = \mathbf{B}\mathbf{u}_{last} + \mathbf{M}_e\mathbf{f}_e^n$
3. $\mathbf{A}_l\mathbf{f}_l = \mathbf{B}_l\mathbf{u} + \mathbf{C}_l\mathbf{u}_{last}$
4. $\phi(\mathbf{f}_{nl}, \mathbf{B}_{nl}\mathbf{u} + \mathbf{C}_{nl}\mathbf{u}_{last}) = \mathbf{0}$
5. $\mathbf{u} := \mathbf{u} + \mathbf{M}_l\mathbf{f}_l + \mathbf{M}_{nl}\mathbf{f}_{nl}$
6. $\mathbf{y}^n = \mathbf{Q}\mathbf{u}$
7. $\mathbf{u}_{last} = \mathbf{u}$

`end`

Such codes have been developed in the style of Music N, requiring, as input, an instrument file, describing the properties of the individual components, connections, as well as multichannel readout locations, and a score file containing information regarding the excitation gesture (in a percussion framework, this is the timing and amplitudes of individual strikes).

The precomputation stage consists of two steps: the generation of matrices from the instrument definition of the modular environment: $\mathbf{A}_l$, $\mathbf{B}_l$, $\mathbf{C}_l$ and $\mathbf{M}_l$, corresponding to linear connections, $\mathbf{B}_{nl}$, $\mathbf{C}_{nl}$ and $\mathbf{M}_l$, corresponding to nonlinear connections and $\mathbf{Q}$, corresponding to readout locations from the modular system. Because of the local nature of finite difference time domain methods, all such matrices are easily constructed, and extremely sparse, and thus storage requirements are low. The second step consists in generating, from score information, an input vector sequence of forces, $\mathbf{f}_e$, and a matrix $\mathbf{M}_e$ (again sparse) from excitation locations. The state $\mathbf{u}_{last}$, is the collection of displacement values over the grids for all components in the system, at two initial time steps, arranged as a single vector; in general, this vector is initialized to zero.

In the run-time loop, step 1., consisting of the construction of sparse matrices $\mathbf{A}$ and $\mathbf{B}$, is necessary only when at least one component in the environment is nonlinear; otherwise, they may be constructed at the precomputation stage—in an explicit update (which is ideal), $\mathbf{A}$ is the identity matrix. Step 2. is a first pass through the system in the absence of connections, and introducing external control signal $\mathbf{f}_e$. Notice that in general, this requires a linear system solution involving $\mathbf{A}$. Step 3. is the calculation of linear connection forces $\mathbf{f}_l$, generally requiring a linear system solution. Step 4. is the calculation of nonlinear connection forces $\mathbf{f}_{nl}$, framed in terms of a nonlinear function ϕ (describing, for example, characteristics of collisions of bowing actions). This step generally requires the use of an iterative method such as Newton-Raphson. Step 5. is the reinsertion of the calculated connection forces into the solution. Step 6. is the calculation of sound output (generally multichannel). Step 7. is the shift of the state in preparation for the next step in the recursion.

The only operations which occur in the run-time loop are sparse matrix-vector multiplication, sparse linear system solution, and iterative methods such as Newton-Raphson. In terms of the raw operation count, steps 1.,3.,5.,6., and 7. are generally not computationally costly. Step 2., if the matrix $\mathbf{A}$ is not the identity (as is the case when nonlinear components are present) can be computationally very heavy, as a large linear system solution is necessary. For

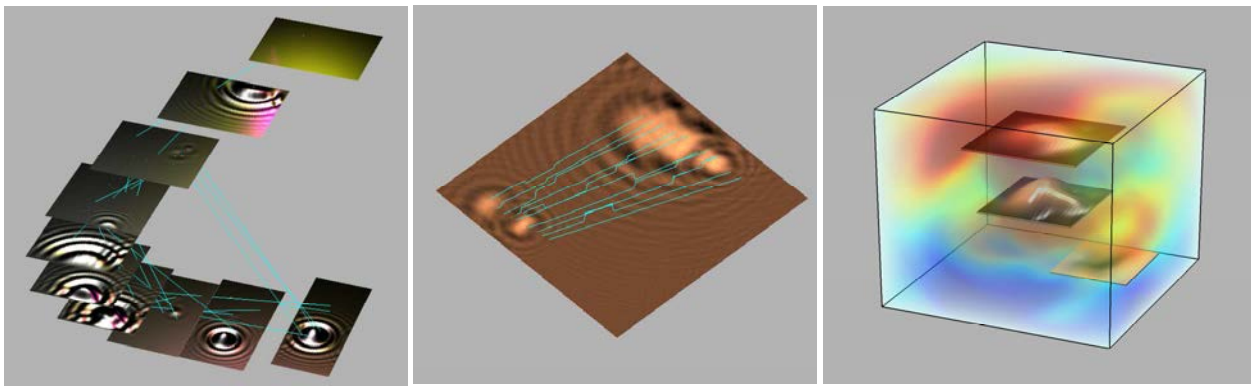

Figure 3. Modular environments: Left: a connected network of plates, as described in Section 4.1. Center: a plate/string network,as described in Section 4.2. Right: a set of nonlinear plates embedded in a 3D air box, as described in Section 4.3.

large 3D systems, even if **A** is the identity, this step can also be extremely costly. Step 4. can also be quite heavy, particularly when regions of distributed nonlinear contact are present (as, for example, in the case of a snare drum model). The implications of these operations for execution time in a parallel implementation will be considered separately in Section 5.

4. ENVIRONMENTS

4.1 A Connected Network of Plates

A rudimentary environment, based on a previous system described in [19], consists of a set of plates, over which the user has individual control over the material type, thickness, dimensions, two-parameter frequency-dependent loss, and boundary condition type (pivoting, clamped, or free); the plates are modelled entirely in 2D, without embedding in a 3D space. See Figure 3, at left. Connections are specified between specified points; such a connection is modelled as a combination of a linear spring, a cubic nonlinear spring, and a linear damper. Input is of three types: striking, consisting of the insertion of pulses of specified duration and amplitude at given locations, bowing, where the user has control over bowing location, force and velocity, and through sending in an audio file, in which case the environment is to be viewed as an effect algorithm. Multichannel output is drawn directly from plate displacement or velocity at specified locations, in a manner somewhat analogous to a contact microphone. Such an environment formed the basis for the investigations of one of the authors (Delap), and the resulting companion piece to this article.

4.2 Plate + Constrained Strings

A different environment, currently under testing, allows for the connection of multiple string or bar modules to a single plate, which could function as a soundboard (if the plate is chosen large and thin enough); such a system is capable of generating sympathetic resonance between the various strings—see Figure 3, at center. In addition, each string is constrained against a barrier (not indicated in the figure), allowing for fretting action and nonlinear effects such as those heard in instruments such as the sitar.

4.3 Embedded Nonlinear Plates in 3D

Another environment based on plate components has been recently developed [20], and is similar in some respects to that presented in Section 4.1, in that the user has control over the different parameters of the system.

In this case, however, the underlying physical model for each plate includes nonlinear effects described by the von Kármán system [21]. At high striking amplitudes, this numerical model produces pitch glides and crashes typical of gongs and cymbals [22], which are perceptually important features that increase the realism of the synthetic sound.

Another difference with the network presented above is that the plates are embedded in 3D, and their position can be specified anywhere within a finite enclosure, as discussed in Section 2.3. See Figure 3, at right. The motion of each plate is coupled with the surrounding air, such that the vibrations produced propagate throughout the finite enclosure—indeed, sympathetic resonance effects between the plate components are a possibility. Output sounds can be picked up simultaneously at multiple points within the box, and a moving output location is also a feasible option. It is also possible to incorporate a cavity terminated on the plate so as to emulate drums.

The excitation mechanism for this environment is still at a preliminary stage, and only short strikes can be fed in at the moment. However, as in the previous case, bowing gestures and input audio files could also be easily implemented.

5. ACCELERATION IN MULTICORE AND ON GPU

Protyping work was carried out in the Matlab language. The hardware platform to which prototype codes were ultimately ported consists of:

- Dual 6 core Xeon E5-2620 CPUs running at 2GHz
- 4 NVIDIA Tesla K20c GPUs

This machine provides a number of opportunities for accelerating the codes by exploiting parallelism at various levels. The GPUs are very well suited to problems where the same calculation is performed across a large number of

Instrument File

```
# zcversion 0
# set 44100Hz sampling rate
samplerate 44100

# define a steel plate
# <name> <material> <thickness> <tension> <X> <Y> <T60@0Hz> <T60@1kHz> <bc type>
plate plat1 steel 0.002 0.0 0.3 0.2 10.0 6.0 4

# define a connection from one point on the plate to another
# <X1 Y1> <X2 Y2> <linear stiffness> <nonlinear stiffness> <T60>
connection plat1 plat1 0.8 0.4 0.6 0.7 10000.0 10000000.0 1000000.0

# define two outputs from the plate <X Y> <pan>
output plat1 0.9 0.6 -1.0
output plat1 0.3 0.7 1.0
```

Score File

```
highpass off # no high-pass filter
duration 1.0 # one second simulation

# define a strike
# <strt T> <component> <X Y> <Duration> <Amplitude>
strike 0.0 plat1 0.4 0.7 0.002 400000.0

# define a bowing action
# <strt T> <comp> <X Y> <dur> <F amp> <V amp> <friction> <ramp T>
bow 0.3 plat1 0.3 0.9 4.0 2.3 2.8 1.1 0.02

# define an audio input
# <file> <strt T> <component> <X Y> <gain>
audio drumming.wav 0.1 plat1 0.2 0.4 1.0
```

Figure 4. Instrument and score files for the modular plate environment described in Section 4.1.

data items and are therefore a good fit for running 3D components, such as the surrounding box described in Section 4.3. They are also effective for speeding up calculation for some of the 2D components (the ones that are linear, as in the environment described in Section 4.1, and reasonably large). 1D components tend not to provide enough parallelism for GPU acceleration to be worthwhile. The GPUs are programmed by using NVIDIA's CUDA toolkit [23] to implement key parts of the code.

There are also two levels of parallelism available on the CPUs. Most obviously, the machine has 12 CPU cores in total, allowing up to 12 threads of execution to run concurrently (though memory bandwidth may become a bottleneck if all of these threads are making heavy use of the shared memory). This can be exploited by using a threading library such as OpenMP [24] or Pthreads [25]. Finally, each CPU core includes a vector unit implementing Intel's SSE (Streaming SIMD Extensions) and AVX (Advanced Vector Extensions) technologies [26]. These allow a single machine instruction to perform multiple calculations simultaneously.

The linear plate network code described in Section 4.1 was accelerated by porting it to the GPUs. For a network consisting of 4 medium sized (100x100 grid points) plates, this gives a 7x speed up over a single-threaded CPU implementation; for 4 large sized (200x200 points) plates, the GPU port is 27x faster.

The plate plus constrained strings environment described in Section 4.2 proved unsuitable for GPU acceleration as the bottleneck is the handling of collisions between the strings and the constraining surface, and the string data is not large enough to make a GPU port worthwhile, even in the most extreme cases (i.e. a large number of low tension strings and a large plate). Instead, an optimised multicore port was created, running the collision code on multiple threads to take advantage of the multiple CPU cores. This gave a speed up of around 60x over the original Matlab version of the code.

The embedded nonlinear plates code described in Section 4.3 was more complex and a hybrid approach was used. The 3D surrounding air box was accelerated with the GPUs, but the 2D plates were unsuitable for GPU acceleration - the linear system solution operation required at each time step uses a preconditioner algorithm that is inherently sequential, and attempts to replace this with a parallel preconditioner were unsuccessful. However, it was possible to optimise the preconditioner on the CPU using vector instructions, and also to take advantage of multiple CPU cores by updating each plate in a separate thread. Finally, the plate updates on the CPU were overlapped with the airbox update on the GPU to reduce the runtime still further. The optimised hybrid code is around 60-80x faster than the original Matlab version of the code.

6. USER INTERFACE

As a preliminary step towards usability, a web-based user interface has been implemented. The interface allows for two main functions: submission of compositions to be run on the dedicated GPU-based hardware, and the generation of percussive gestures to be used in a composition score.

As of March 2014, there are three different modular codes available via the user interface for the composers, representing the environments as discussed in Section 4.1, 4.2

and 4.3. The simulations (hybridized over the GPU and host in multicore, as discussed in the previous section) that can be run with these codes are controlled by two input files: the instrument definition file, and the score file. See Figure 4 for a simple example of an instrument/score file pair. The instrument file describes the individual properties of a series of plates, as well as connections among them, their type, and output locations. The score file describes either a series of gestures (such as strikes, bowing actions), or an input audio file, to be used as excitations for the instruments.

Given that a score could ultimately be quite lengthy, in order to quickly create a list of gestures a simple gesture generator web interface was developed. This allows the composer to quickly generate a series of strikes based on a user-defined profile, controlled by a series of breakpoint functions (see Figure 5). These can then be used in a rapid-calculation demo mode on a basic steel plate, or exported directly to form part of a score. gestural properties which can be specified are strike density (in strikes per second), amplitude, duration, and x and y location of the strikes on the plate as a function of time. A user-controlled randomizing function is also incorporated.

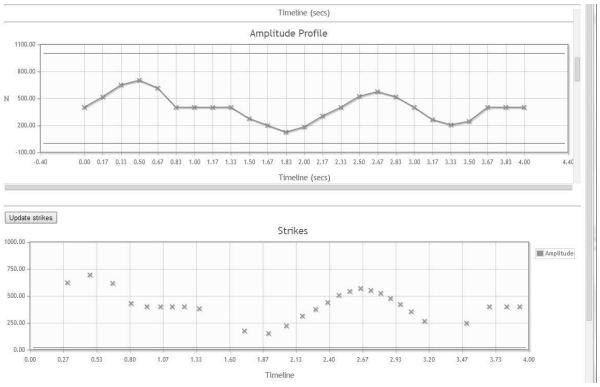

Figure 5. The user interface gesture generator for the modular plate network.

Further details on the user interface are available at:
`http://www.ness-music.eu/user-documentation`
.

7. A COMPOSER'S PERSPECTIVE

The multichannel fixed-media composition, *Ashes to Ashes*, was generated mainly through compositional exploration of the modular environments described in Section 4.1 and Section 4.3. One other modular physical modeling synthesis instrument, not described here, but currently in an intermediate stage of porting from Matlab was also employed: a multi-valve brass instrument, also availed of and fused into the fabric of the work.

The sound artist adopts multiple roles in terms of dealing with materials and gestures: instrument builder, performer, and composer. A high level of subtlety can be required in voicing instruments subsequent to design. The connected plate network is particularly suited to creating percussion instrument archetypes. However, through extreme instrument configurations, it is possible to generate sonic outputs which might be unrealizable under natural conditions as a consequence of an infeasible design, yet which maintain the authenticity of the percussion-type instruments. Atypical outcomes can also be realized by driving plate models in an "effect" mode with audio files. Investigation of such a spectrum of instruments with familiar instrument types positioned at one end, and novel sound material located at the other presented one avenue of creative inquiry.

The user interface allowed for the generation of gestures of considerable complexity. Such gestures could imply a high degree of human agency (e.g. drumrolls) or could imply gestures of non-human origins (machinery, rain). Interplay between instrument and gesture types led to the creation of three soundworlds within the composition:

- A soundworld whereby human agency is strongly implied through excitation of more-or-less familiar instrument types (gongs, cymbals, bass drums, bells) and integrating recorded speech elements which were driven through large metal plates. Wind instruments, generated through the wind instrument models, and passed through very large plates were used in this section.

- A soundworld where instrument types were less familiar and where human agency was masked or subverted. A large network of plates was excited in a mechanistic fashion, and was combined with enormous wind instruments which had been driven through metal plates. Breath sounds associated with the wind instrument models also featured heavily here.

- A soundworld dealing with unfamiliar objects, and one in which cues indicative of human performance are largely absent. In this case, sound objects were provided with dimensions which are highly unusual in the design of typical instruments, and were excited though gestures abstracted from standard modes of human performance.

A materials component is present which allowed for specification of parameters pertaining to the physical properties of specific metals—in this case, uranium. The ability to allude to the properties of a particular metal was important on a conceptual level.

8. CONCLUDING REMARKS

The NESS Project is concerned with raw sound generation from physical models, and in particular the algorithmic and computational aspects of synthesis. The musical goal is, simply speaking, to explore the possibilities for synthetic sound, while making as few simplifying assumptions as possible about the systems under consideration. Even for relatively large scale systems, implementation in parallel hardware can (but does not always!) lead to great acceleration—synthesis is not real time, but not as far

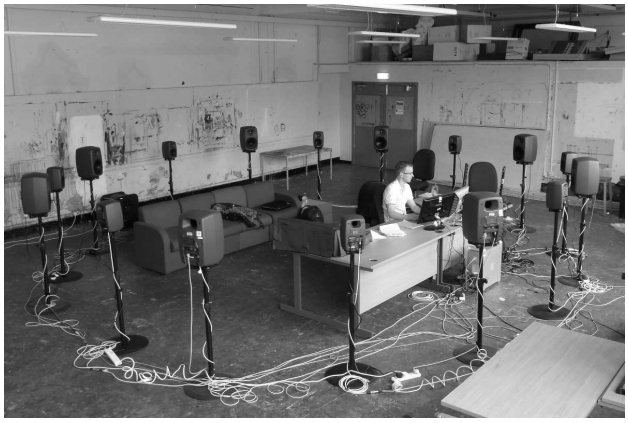

Figure 6. Composer at work in the makeshift 16-channel space at the University of Edinburgh.

off as one might think; though execution time depends on the complexity of the instrument, sound generation times for can often be faster than real time, even for relatively complex forms. The constraints of parallel implementation have informed many aspects of algorithm design here, and in particular the use of regular grids whenever possible. Some necessary computational operations, such as large linear system solutions and root-finding, can be difficult to parallelize using standard solution methods, and the search for parallelizable alternatives forms a major part of ongoing algorithm design work.

The larger question, and not one addressed under the current project, is that of user instrument design and control. At the level of instrument design, individual modules are characterised by a small number of parameters. When there are many such modules (as in the environment described in Section 4.1), the parameter space becomes large, and some means of exploring such a space becomes necessary. At the same time, such exploration forms part of the learning experience of the musician, who quickly deduces general constraints on the design space to the "musically useful," much in the same way as an instrument builder. While in the current non-real-time setting, a playable control interface is clearly not a concern, interesting percussive gestures can require a large number of individual events, and the simple strategy described in Section 6 allows for low-dimensional control over such gestures.

The ultimate answers to questions regarding instrument design, control and musical use, however, can only be arrived at through close work with musicians—this collaboration has been the first such exploration, and is one of a further five to be carried out over the next three years.

Acknowledgments

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

9. REFERENCES

[1] C. Cadoz, "Synthèse sonore par simulation de mécanismes vibratoires," 1979, thèse de Docteur Ingénieur, I.N.P.G. Grenoble, France.

[2] 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.

[3] J.-M. Adrien and X. Rodet, "Physical models of instruments, a modular approach, application to strings," in *Proceedings of the International Computer Music Conference*, Vancouver, Canada, 1985, pp. 85–89.

[4] J.-M. Adrien, "The missing link: Modal synthesis," in *Representations of Musical Signals*, G. DePoli, A. Picialli, and C. Roads, Eds. Cambridge, Massachusetts: MIT Press, 1991, pp. 269–297.

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

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

[7] 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.

[8] A. Fettweis, "Wave digital filters: Theory and practice," *Proceedings of the IEEE*, vol. 74, no. 2, pp. 270–327, 1986.

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

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

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

[12] C. Vallette, "The mechanics of vibrating strings," in *Mechanics of Musical Instruments*, A. Hirschberg, J. Kergomard, and G. Weinreich, Eds. New York, New York: Springer, 1995, pp. 116–183.

[13] V. Välimäki, T. Tolonen, and M. Karjalainen, "Plucked-string synthesis algorithms with tension modulation nonlinearity," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, Phoenix, Arizona, March 1999, pp. 977–980.

[14] B. Bank and L. Sujbert, "Generation of longitudinal vibrations in piano strings: From physics to sound synthesis," *Journal of the Acoustical Society of America*, vol. 117, no. 4, pp. 539–557, 2005.

[15] S. Bilbao, "Robust physical modeling sound synthesis for nonlinear systems," *IEEE Signal Processing Magazine*, vol. 24, no. 2, pp. 32–41, 2007.

[16] 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.

[17] S. Bilbao, "Time domain simulation of the snare drum," *Journal of the Acoustical Society of America*, vol. 131, no. 1, pp. 913–925, 2012.

[18] S. Bilbao, B. Hamilton, A. Torin, C. Webb, P. Graham, A. Gray, J. Perry, and K. Kavoussanakis, "Large scale physical modeling synthesis," in *Proceedings of the Sound and Music Computing Conference*, Stockholm, Sweden, 2013.

[19] S. Bilbao, "A modular physical modeling synthesis environment," in *Proceedings of the International Digital Audio Effects Conference*, Como, Italy, September 2009.

[20] A. Torin and S. Bilbao, "A 3D Multi-Plate environment for sound synthesis," in *Proc. of the 16th Int. Conference on Digital Audio Effects (DAFx-13)*, Maynooth, Ireland, September 2-6, 2013.

[21] A. H. Nayfeh and D. T. Mook, *Nonlinear oscillations*. New York: John Wiley and Sons, 1979.

[22] A. Chaigne, C. Touzé, and O. Thomas, "Nonlinear vibrations and chaos in gongs and cymbals," *Acoustical science and technology*, vol. 26, no. 5, pp. 403–409, 2005.

[23] "Cuda parallel computing platform," http://www.nvidia.co.uk/object/cuda_home_new.html, accessed: 2014-03-28.

[24] L. Dagum and R. Menon, "Openmp: an industry-standard api for shared-memory programming," *IEEE Computational Science and Engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[25] F. Müller, "A library implementation of posix threads under unix," *InUSENIX*, p. 2941, January 1993.

[26] N. Firasta, M. Buxton, P. Jinbo, K. Nasri, and S. Kuo, "Intel AVX: New frontiers in performance improvements and energy efficiency," Intel white paper, 2008.