Audio Engineering Society

Convention Paper

Presented at the 128th Convention
2010 May 22–25 London, UK

Finite Difference Room Acoustic Modelling on a General Purpose Graphics Processing Unit

Alex Southern[1,2], Damian Murphy[1], Guilherme Campos[3], Paulo Dias[3]

[1] *Audio Lab, Department of Electronics, University of York, York, YO10 5DD, UK*

[2] *Department of Media Technology, Aalto University School of Science and Technology, PO Box 15400, FI-00076, Finland*

[3] *IEETA/Department of Electronics, Telecommunications and Informatics, University of Aveiro, 3810-193 Aveiro, Portugal*

Correspondence should be addressed to Alex Southern (aps502@ohm.york.ac.uk)

ABSTRACT

Detailed and convincing walkthrough auralizations of virtual rooms requires much processing capability. One method of reducing this requirement is to pre-calculate a data-set of room impulse responses (RIR) at locations throughout the space. Processing resources may then focus on RIR interpolation and convolution using the dataset as the virtual listening position changes in real-time. Recent work identified the suitability of wave-based models over traditional ray-based approaches for walkthrough auralization. Despite the computational saving of wave-based methods to generate the RIR dataset, processing times are still long. This paper presents a wave-based implementation for execution on a general purpose graphics processing unit. Results validate the approach and show that parallelisation provides a notable acceleration.

1. INTRODUCTION

Highly accurate soundfield rendering can be achieved by first accurately predicting the nature of wave propagation throughout an arbitrarily shaped multi-dimensional enclosure such as a surface or volume and has been the subject of much research interest. These different techniques can be divided into two different categories; ray-based and wave-based modelling techniques. Each of these fundamental categories has related advantages and limitations but one key difference has been the relative computational intensity of wave-based methods par-

ticularly if the complete audio spectrum is desired.

This paper focuses on accelerating wave-based room acoustical modelling by specifically implementing a finite difference time domain (FDTD) scheme over an array of multiprocessors using a general purpose graphics processing unit (GPGPU).

2. ROOM ACOUSTIC MODELLING

There is a fundamental difference between ray-based and wave-based acoustical models. Ray-based methods make the assumption that sound propagates in a specular ray-like manner that exhibits specular reflection characteristics at boundaries. Wave-based methods take regard for the physical wave motion of sound propagation by forming approximate discrete numerical solutions to the wave equation. This section gives a brief overview of some popular acoustical modelling strategies and gives particular focus to the FDTD method used in this work.

2.1. Ray-Based Methods

Ray-based (or *geometric*) methods are the underlying acoustic modelling techniques that are used in commercial applications such as CATT-Acoustic [1], ODEON [2], Ramsete [3] and EASE [4]. These techniques work on the principle that a Room Impulse Response (RIR) can be synthesized if the time of arrival from source to receiver (virtual microphone) for the numerous propagation paths, or *rays*, that exist for an arbitrary source/receiver/geometry combination are known. The difference between different ray-based models such as the image-source and ray tracing methods is in the approach for calculating the various ray-paths for an arbitrary geometry. These approaches fail to properly account for phase information meaning that wave interference effects are not inherently modelled and this is particularly important for the validity of the low frequency region of the RIR [5]. These techniques are not the focus of this work and so the reader is referred to [6][7].

2.2. Wave-Based Methods

Wave-based acoustic simulations offer greater accuracy in the low frequency bands and can model wave phenomena including wave interference, occlusion and diffraction inherently. These approaches include the digital waveguide mesh approach (DWM) [8][9], functional transformation method (FTM) [10][11] and the FDTD method afore mentioned [12][13].

The DWM is based on an extension of the Digital Waveguide (DWG) commonly used in musical acoustics applications to physically model a 1D system such as a resonating tube or vibrating string [14]. The DWM is now well established in the literature where its derivation, application and shortcomings are well documented [8][9]. In FTM an analytic frequency domain solution of an arbitrary linear partial differential equation (PDE) is found that describes a continuous vibrating system such as a string or membrane. The solution is then transformed back into the time and space domain to be discretely implemented. In this work a FDTD method is of primary interest and is therefore introduced in more detail.

2.3. The FDTD method

FDTD models are well established for the purpose of approximately simulating the wave propagation of sound throughout an enclosed space over a period of time [12][13]. The 2D or 3D wave equation is discretized using a second order central finite difference approximation to the wave equation. Finite difference equations for dealing with air, boundaries and corners can be formulated along with the addition of a spatio-temporal indexing notation for convenience as follows [15]:

$$p_{i,j}^{n+1} = \lambda^2(p_{i+1,j}^n, p_{i-1,j}^n, p_{i,j+1}^n, p_{i,j-1}^n) + 2(1-2\lambda^2) - p_{i,j}^{n-1} \quad (1)$$

$$p_{i,j}^{n+1} = [2\lambda^2 p_A^n + \lambda^2(p_{B1}^n + p_{B2}^n)(\frac{\lambda}{\xi_w} - 1)p_{i,j}^{n-1} + 2(1-2\lambda^2)p_{i,j}^n]/(1+\frac{\lambda}{\xi_w}) \quad (2)$$

$$p_{i,j}^{n+1} = [2\lambda^2(p_{B1}^n + p_{B2}^n) + (\frac{2\lambda}{\xi_w} - 1)p_{i,j}^{n-1} + 2(1-2\lambda^2)p_{i,j}^n]/(1+\frac{2\lambda}{\xi_w}) \quad (3)$$

where (1),(2) and (3) are the 2D air, boundary and corner update equations respectively. A, $B1$ and $B2$ are the ij co-ordinates of the neighbouring air node and boundary nodes. $\lambda = 1/\sqrt{2}$ is the courant number. Finally $\xi_w = Z_w/\rho c$ and is the normalized wall impedance where ρ is air density, c is speed of sound and Z_w is the chosen boundary impedance [15]. The update equations are used to iteratively calculate the pressure values for each time step n over a regularly spaced grid of points called *nodes* that discretely represent a room geometry as depicted for 2D in Figure

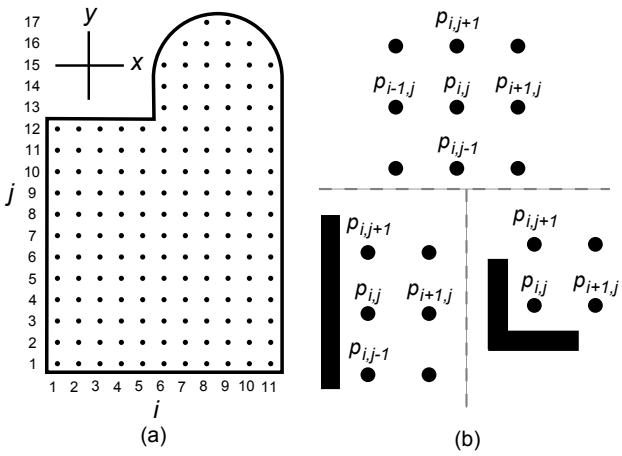

Fig. 1: (a) An illustrative example of a discretely spaced grid of nodes used to represent an arbitrary enclosed geometry. (b) The index notation used to reference a node and its corresponding neighbours.

```
READ 2D Array of Node Types
READ simDuration
READ width of Grid to X
READ length of Grid to Y

FOR n = 1 TO simDuration

 FOR i = 1 TO X
  FOR j = 1 TO Y

   CASE P(i,j) OF
    airNode:
      COMPUTE new air node pressure at [i,j]
    boundaryNode:
      COMPUTE new boundary node pressure at [i,j]
    cornerNode:
      COMPUTE new corner node pressure at [i,j]
    OTHERS:
      COMPUTE nothing
   ENDCASE
  ENDFOR
 ENDFOR

 WRITE pressure at receiver to memory
ENDFOR
```

Fig. 2: A pseudocode listing outlining a simple 2D FDTD implementation.

1(a). In Figure 1(b) the index notation for the three different update equations is demonstrated for specific boundary and corner orientations.

The FDTD scheme offers greater RIR validity at lower frequencies when compared to ray-based approaches as wave phenomena such as diffraction and occlusion are inherently modelled. Accuracy at high frequencies is available when the inter-nodal distance (4) is chosen to be sufficiently small, at the expense of larger computation times and memory requirements.

$$dx = \frac{\sqrt{D} \cdot c}{fs} \tag{4}$$

where dx is the inter-nodal distance in metres, D is model dimensionality, c is the speed of sound and fs is the chosen sampling rate.

Using (1),(2) and (3) it is possible to physically model the RIRs of arbitrary 2D representations of complex room geometries. A simple implementation overview is provided in Figure 2. Applying this code listing to the mesh geometry in Figure 1(a), X = 11 and Y = 17. The algorithm systematically, on a sequential node-by-node basis, calculates the pressure at each node for each time step. If a node is flagged as a receiver then each pressure value at each time step is written to an audio file. A comprehensive research tool for physically modelling room acoustics was presented in [16].

This paper focuses on accelerating the FDTD method for acoustic modelling by calculating the pressure at multiple nodes simultaneously.

3. AURALIZATION

The underlying aim of this work is to provide an aural presentation of a physically modelled soundfield to a listener. The aural presentation of a modelled soundfield is more commonly referred to as *auralization* [17]. It is also desirable to have dynamic auralization systems that allow the listener to virtually navigate or *walkthrough* the modelled enclosure. The modelling techniques chosen to achieve this depend on the context of the implementation, however due to the computational constraints of wave-based methods, the ray-based methods have proven most successful [18][19][20].

In [18] the authors calculate the first order reflections in real-time using the image-source method while the reverberation tail was generated using a general recursive delay network. Therefore this style of approach trades RIR integrity for reduced computational intensity. Another approach that avoids compromising the RIR accuracy involves pre-computing an RIR data-set for different source/receiver combi-

nations. Dynamic real-time walkthrough auralizations are then possible if the appropriate RIRs for the virtual listener location are convolved and interpolated using the data-set [19][20]. The interpolation from both measured and synthesized data-sets were considered in [21] and [22]. A disadvantage of using such data-sets is apparent when both source and receiver locations are required to dynamically change at run-time. In this case a set of data-sets is required.

A possible approach for the implementation of a wave-based walkthrough auralization system has been presented in [23]. It is pointed out that for wave-based methods it is necessary to update all points throughout the grid in order to calculate a single RIR. In this scenario the ray-based methods are more efficient. However if the RIR at all points in the grid are desired for a single source layout then the wave-based methods incur no additional computation time. This therefore makes wave-based methods more efficient at generating large RIR datasets suitable for dynamic walkthrough auralizations.

In summary, dynamic auralization by real-time RIR synthesis using wave-based methods is currently not possible on standard desktop hardware. However by accelerating the FDTD method the time taken to generate multiple RIR data-sets can be further reduced when compared to the same task using ray-based methods.

4. FDTD FOR GRAPHIC PROCESSING UNITS

In order to move towards the possibility of highly accurate virtual auditory walkthrough systems it is necessary to perform accurate acoustic modelling of the desired enclosed environments. To this end a parallel implementation of a 2D FDTD acoustical model is highly beneficial. In recent years General Purpose Graphics Processing Units (GPGPU) have become more affordable and accessible to the casual programmer. GPGPUs are capable of performing many instructions in parallel over an array of multiprocessors and offer potential optimizations when the computational task is suitably arranged for the device. Parallel solutions to the problem of accurately predicting the nature of wave propagation throughout an arbitrarily shaped multi-dimensional enclosure, such as a surface or volume have been considered previously and are briefly discussed in the following.

Ray-based modelling techniques have been the first strategies to benefit from the parallel computation capabilities offered by the GPU. This is primarily because of the inherent similarities that exist between the computer graphic shading algorithms and acoustic ray-tracing. Freely available graphics APIs, such as OpenGL, have enabled the programmer to implement acoustic ray-tracing on a GPU without the need for a detailed understanding of the device architecture, although an appreciation of the graphics terminology is essential. The possibility of utilizing GPUs for wave-based calculations has been explored by a number of authors previously in seismic analysis [24][25] as well as electromagnetics [26, 27, 28, 29, 30, 31]. The application of GPU devices for the specific task of wave-based numerical approximation of room acoustics was proposed in [32]. The author presents a DWM implementation on a GPU that employs a shader language to perform the propagation step on the acoustic pressure values stored as RGB textures. An approach presented in [33] used a software based central work flow co-ordinator called Parallel Virtual Machine (PVM) to perform parallel computations by treating multiple PC's on a network as one computer.

The compute unified device architecture (CUDA) [34] delivers parallel capabilities of the GPU to the wider non-graphics programming communities and was employed in [24, 25, 29, 31]. A CUDA implementation is presented in this paper with application to room acoustic modelling.

4.1. Compute Unified Device Architecture

CUDA offers the multi-core processing power of a GPU through an extended C library that is inherently low level and independent of graphics terminology. The program flow of an arbitrary CUDA implementation is illustrated in Figure 3. The program is invoked on the CPU, or host machine, by the user. The host allocates memory on both the host and GPU device then reads in any required data and copies the relevant areas of host memory onto the device where necessary. A user defined function called a *kernel* is then called on the GPU to perform concurrent instruction execution on the data. The kernel is executed multiple times by the same number of *threads*, each thread instance has its own unique thread ID co-ordinate within a thread *block*. Each block has a unique block ID co-ordinate within a

grid. Upon completion of the kernel phase, the host performs any necessary memory copies from device to host. Multiple calls to different kernels performing specific parallel tasks on the same data can be made if required.

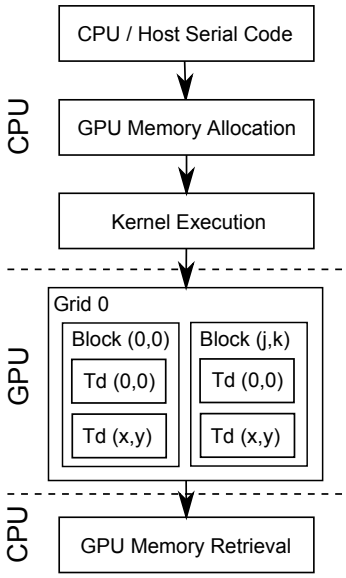

Fig. 3: The program flow of a typical CUDA application. Tasks that are data parallel are executed on the GPU using a kernel function.

Threads within a thread block are able to communicate via the shared memory. Communication between blocks is only possible via the global memory space. The importance of this to the programmer is that the time taken to read/write to and from shared or global memory spaces differs greatly and therefore care must taken to minimize certain read/write types. Specifically, reading/writing data from the global memory space is reported to be 100 to 150 times slower than reading/writing to shared memory spaces [34]. However, the off-chip global memory space is much larger than the on-chip shared memory and so utilizing both areas with minimal reading/writing to global memory is a key optimization opportunity. A direct result of these device constraints is that there are a number of considerations that must be addressed by the programmer when attempting to implement a parallel solution on a CUDA enabled device.

1. Memory transfers from host to device and device to host should be minimal.

   These transfer times can be relatively long when there is a comparatively large amount of data. When necessary, host to device memory transfers can be asynchronous with host CPU functions.

2. Thread blocks set up by any one kernel function must be executable in any order.

   Ensuring that blocks can be executed in any order allows a CUDA program to be inherently scaled at run-time for execution on different GPU specifications.

3. Global memory access performed by a kernel should occur in an organized manner.

   When a number of data values in the global memory space can be accessed in an organized manner the device is able to perform memory *coalescing*, that is, multiple loads from global memory to shared memory and vice versa can be performed in one operation.

4. *Bank conflicts* cause operation serialization and should be avoided.

   Bank conflicts occur when multiple data requests to shared memory result in simultaneous access to the same *memory bank*. Shared memory is made up of multiple memory banks which may be accessed concurrently, otherwise the requests are serialized.

4.1.1. Memory Coalescing

Under certain conditions a group of 16 threads simultaneously loading/storing data between shared and global memory can complete the transaction in just one operation as opposed to separate serialized transactions.

The global memory space should be utilized so that data is organized into a listing of 64 byte or 128 byte words. For example, 1 single floating point value is 4 bytes (32 bits) in size. Therefore 16 floating point values can be stored into a 64 byte word.

When reading values from the global memory space, for coalescing to occur it is necessary to have a group of 16 threads (known as a *half warp*) access the sequentially corresponding 4 byte word in the segment.

When these conditions are met the full 64 byte word can be transferred in a single operatation (or two operations for a 128 byte word).

Therefore in order to successfully accelerate the FDTD method for room acoustic modelling the proposed implementation should be designed with these considerations in mind.

5. A PARALLEL IMPLEMENTATION

The update equations (1),(2) and (3) show the requirement to store the pressure of every node for the past and current time steps. Therefore in the GPU device the number of bytes required to perform the model is given in (5). The future time step overwrites the past time step and so only two pressure fields are required.

$$N_{bytes} = 2 \cdot F_{size} \cdot N_{nodes} \tag{5}$$

where N_{bytes} = number of bytes required to perform the model. The number of bytes used to represent each pressure value, i.e. for a 32-bit single float $F_{size} = 4$. N_{nodes} = number of nodes in the mesh. Therefore for a practical acoustical simulation the pressure values must be stored in the large yet slow access global memory space.

With this in mind, the mesh geometry is broken down into a number of 16x16 sub-grids of nodes called *tiles* that are each processed by a 16x16 thread block in parallel on the GPU. The dimension is chosen as 16 because this is the size of a half warp and therefore 16 single floating points values form a 64-byte word that preserve the memory alignment consideration and therefore meets the coalescing conditions. The FDTD formulations (1),(2) and (3) inherently require that an implementation must perform multiple accesses to the same memory location within the same time step. As all the pressure values must be stored in global memory the repeated memory accesses are undesirable due to the slow access times. To alleviate this issue each tile is copied to the faster shared memory space on the GPU in a coalesced fashion therefore reducing the number of operations required in the tile transfer. The tile update is then performed by accessing the copied pressure values in the shared memory space as the repeated memory accesses are *hidden* by the much faster access times. The updated tile is then copied in a coalesced manner back to global memory overwriting the now redundant past pressure values. Making this design choice does raise some problems which are described here with the aid of Figure 4. For example, assuming they are all air nodes it is possible to calculate the future pressure values for the inner 14x14 grid of any tile. Although, the nodes shaded darkest grey are marked as non-calculable as additional loads from adjacent tile(s) are required to calculate the future pressure value at their location. This is because it is not possible for threads from different blocks to directly share data as there is often more blocks than available processors. The missing neighbour information for each block must instead be retrieved from the global memory. The row above or below a tile can be loaded in a coalesced manner because the 16 elements of a tile row appear in sequential order in memory. This is not true of the adjacent columns to a tile where the values appear in 64 byte word intervals, in this case coalescing cannot occur and these nodes are referred to as *remaining nodes*, see Figure 5. Tiles on the vertical edges of the tile grid only have 1 column

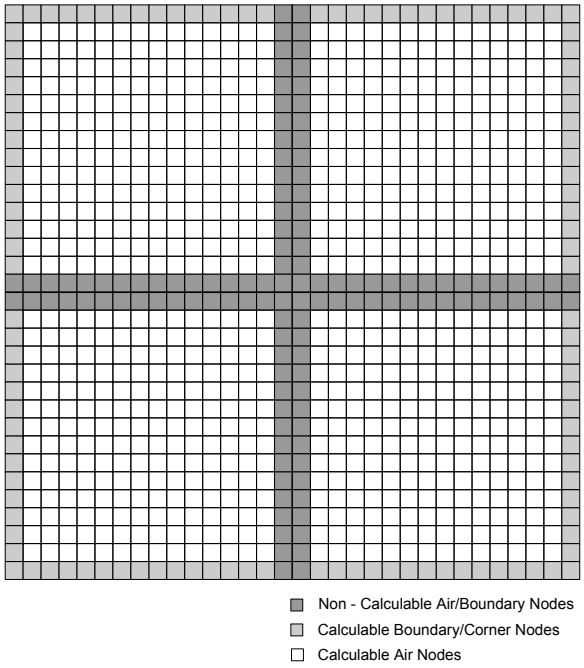

Fig. 4: A 32x32 mesh divided into 4 16x16 tiles. For some nodes it is not possible to calculate the new future pressure values.

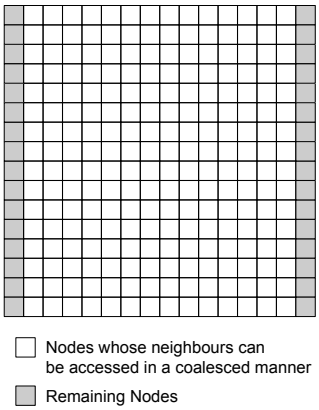

Fig. 5: A central tile with the remaining nodes on both vertical sides highlighted in grey.

of remaining nodes as the other side are boundary nodes. In this implementation we define multiple types of tile each with their unique tile type index (TTI) shown in Figure 6. These 12 tile types are used to represent a complex geometry. This discrimination means that every valid tile can be loaded by a thread block into shared memory in a coalesced fashion and 256 (16x16) individual node type checks do not need to be performed as the tile type infers a particular node type arrangement. There may also be invalid/unused, TTI = 0, tiles which can be skipped again meaning 256 node type interrogations can be avoided at run-time. As some geometries may not be equally divided by tiles there will be some nodes that require updating in a separate manner, these nodes will also be classed as remaining nodes.

The division of an arbitrary mesh into tiles and identification of the two different types of remaining

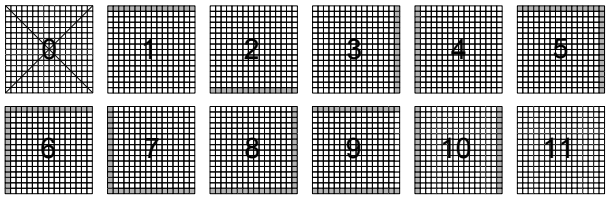

Fig. 6: Definition of the fundamental tile types and their index values. The dark nodes are boundary/corner nodes, white nodes are air nodes. Tile 0 contains no active nodes and is therefore an invalid tile

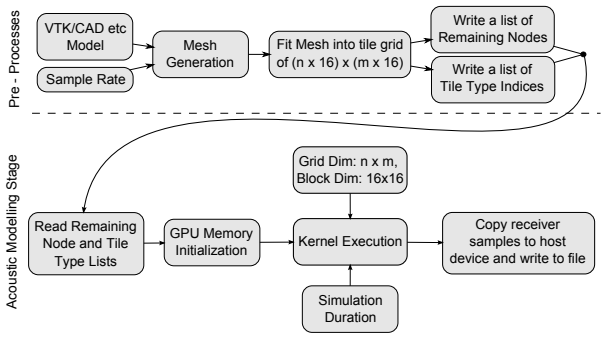

Fig. 7: A top level overview of the proposed GPU implementation.

nodes that cannot be updated in a coalesced fashion are performed as pre-processes to the main acoustical modelling stage. Figure 7 gives an overview of the complete proposed process starting from a geometrical definition of a room model through to the generation of an RIR.

At the kernel stage, the TTI kernel updates all tiles and all the nodes within them, apart from the remaining nodes. The remaining node list (RNL) kernel updates all the nodes listed as remaining nodes in the pre-processing stage. The process is repeated for each time step for the number of specified time steps. The flow diagram for the TTI and RNL kernels is given in Figure 8 and Figure 9.

6. TESTING AND RESULTS

In order to assess the suitability of the proposed implementation for accelerating 2D FDTD room acoustic modelling it is necessary to compare the performance against a typical CPU implementation. In this case the CPU is a Intel Core 2 Duo 3.16GHz running Windows XP Professional with 3.25 GB of RAM. The time taken to calculate 44100 samples of an RIR is measured to the nearest second using the standard timing functions in C for a number of common CUDA enabled GPUs. The comparison is made over 8 different test mesh sizes starting from 32^2, as shown Figure 4, up to 4096^2. The remaining node and tile type lists are generated for each test mesh prior to the modelling stage. The results are given in Table 1 and plotted in Figure 11. Note that for simulations taking longer than 3 hours, the simulation was terminated. Each GPU was also the main graphics card for the operating system. The

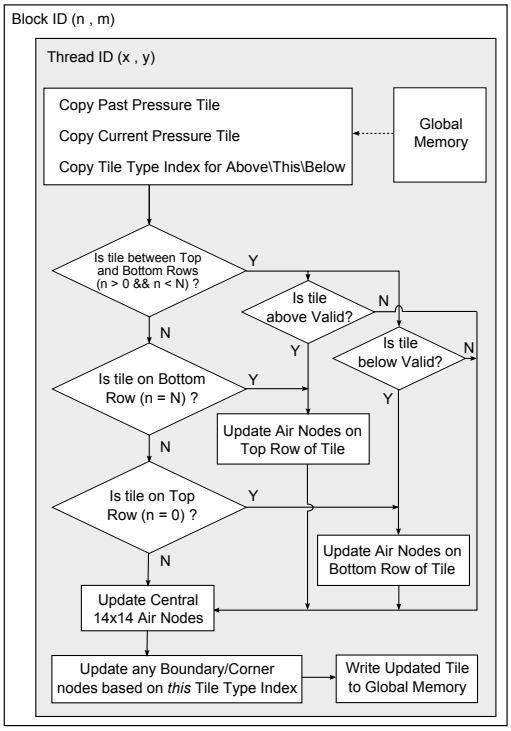

Fig. 8: The flow diagram overview for the TTI kernel

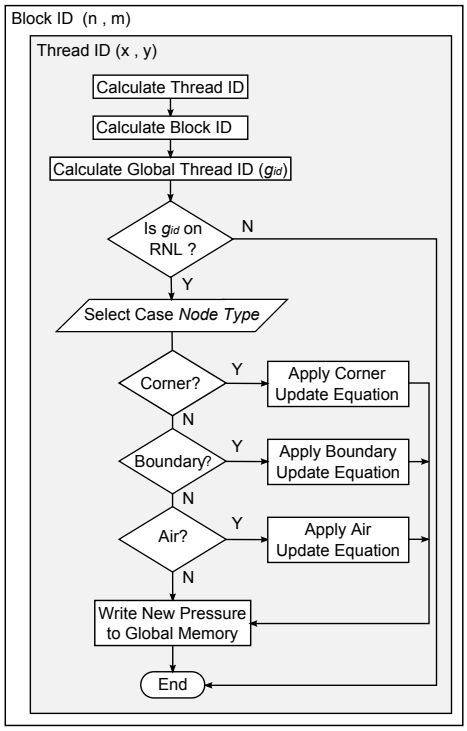

Fig. 9: The flow diagram overview for the RNL kernel. The g_{id} is the global thread ID that identifies the thread within the entire grid.

size of the global memory space on the GPUs varied and so some devices ran out of memory. This was due to the use of an unoptimised list of nodes indices, their associated neighbour indices and their node type. Some global memory could be regained by only storing the indices of the remaining nodes. While this does mean that devices prematurely ran out of memory it does not however affect the processing times.

The varying computation times of the GPU tests are down to the differing hardware specifications and in particular the number of processing cores. The implementation here is scalable for the different GPUs and will utilize all available processors when the computational task is large enough. The scalability is generally inherent in all CUDA implementations due the requirement that all blocks should be executable in any order, although some CUDA functions are specific down to a particular GPU CUDA generation.

6.1. An Additional Test

The proposed method is capable of modelling complex geometries. A complex test geometry is presented in Figure 10 and cannot be divided equally into tiles due to the curved areas. Those tiles that are only part inside the geometry cannot be assigned a TTI. All the active nodes within the tile are therefore added to the RNL, these areas are shaded in black. There are also areas in the geometry that produce discontinuities in the main space, such as a supporting pillar in a hall. Each square within Figure 10 represents the 16x16 tile of nodes.

The complex model was executed on both the standard CPU and on the GeForce GTS 250. The resulting RIRs were identical and the timing results show that the GPU implementation is still considerably more efficient than the CPU implementation.

7. CONCLUSIONS AND FUTURE WORK

This paper has discussed in some detail an accel-

Processor Type	Size of Test Mesh (in Nodes)							
	32^2	64^2	128^2	256^2	512^2	1024^2	2048^2	4096^2
Test Number	1	2	3	4	5	6	7	8
Intel Core 2 Duo 3.16GHz	< 1	2	9	195	810	3323	TL	TL
GeForce GT 8600M	2	6	17	52	172	711	3245	NEM
GeForce GT 8600	1	3	12	43	156	656	NEM	NEM
GeForce GT 8800	1	2	7	25	86	290	1390	NEM
GeForce GTS 250	1	2	6	20	69	234	1130	5404
GeForce GTX 275	1	1	2	3	12	45	179	NEM

Table 1: The computation times for each test case on each processing device rounded to the nearest second. NEM is not enough memory, TL is too long, the simulation went on for over 3 hours

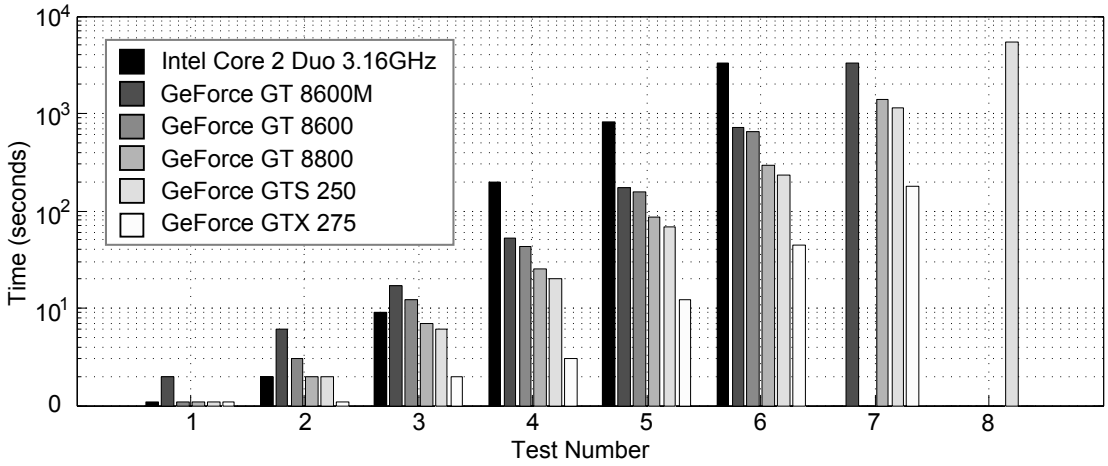

Fig. 11: Bar graph of the computation time in seconds versus the test number. Note that time in seconds is given in a logarithmic scale.

erated GPU CUDA implementation of a 2D FDTD acoustical model for generating RIR responses of arbitrary room geometries. Significant accelerations have been demonstrated for numerous GPUs when compared to a typical 2D FDTD CPU implementation. The results further re-enforce the suitability of the wave-based techniques for generating RIR datasets for detailed walkthrough auralizations.

Future work should include a comparison with the GPU implementation proposed in [35] and the possibility of using other available device memory spaces. The proposed approach may be extended to 3D, however memory size will constrain the implementation to narrow bandwidth and/or small enclosures.

8. REFERENCES

[1] "Catt-acoustic," Online at http://www.catt.se.

[2] "Odeon - room acoustics software," Online at http://www.odeon.dk/.

[3] "Ramsete - room acoustics modeling on pc," Online at http://www.ramsete.com/.

[4] A. D. Ahnert, "Ease," Online at http://www.ada-acousticdesign.de/set_en/setsoft.html.

[5] A. Krokstad, S. Strom, and S. Sørsdal, "Calculating the acoustical room response by the use of a ray tracing technique," *Journal of Sound and Vibration*, vol. 8, no. 1, pp. 118 – 125, 1968.

[6] M. Noisternig, B. Katz, S. Siltanen, and L. Savioja, "Framework for real-time auralization in architectural acoustics," *Acta Acustica united with Acustica*, vol. 94, no. 6, pp. 1000 – 1015, 2008.

[7] P. Svensson, "Modelling acoustic spaces for audio virtual reality," in *Proc. 1st IEEE Benelux Workshop on Model based Processing and Coding of Audio (MPCA-2002), Leuven, Belgium*, November 2002.

[8] S. A. Van Duyne and J. O. Smith, "Physical modeling with the 2-d digital waveguide mesh," in *Proc. Int. Computer Music Conf., Tokyo, Japan*, 1993, pp. 40–47.

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

[10] L. Trautmann and R. Rabenstein, "Digital sound synthesis based on transfer function models," in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, New Paltz, NY, USA*, 1999.

[11] S. Petrausch and R. Rabenstein, "Wavefield simulation with the functional transformation method," in *International Conference on Audio, Speech and Signal processing, Toulouse, France*, 2006.

[12] 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.

[13] K. Kowalczyk and M. van Walstijn, "Formulation of a locally reacting wall in finite difference modelling of acoustic spaces," in *Proc. of the International Symposium on Room Acoustics, Seville, Spain*, 2006.

[14] J. O. Smith, "Physical modeling using digital waveguides," *Computer Music Journal*, vol. 16, no. 4, pp. 74–87, 1992.

[15] M. Noisternig, B. Katz, S. Siltanen, and L. Savioja, "Formulation of locally reacting surfaces in fdtd/k-dwm modelling of acoustic spaces," *Acta Acustica united with Acustica*, vol. 94, pp. 891 – 906, 2008.

[16] M. Beeson, A. Moore, D. Murphy, S. Shelley, and A. Southern, "Renderair - room acoustics simulation using a hybrid digital waveguide mesh approach," in *Proc. of the AES 124th Convention: New Horizons in Audio, Amsterdam, The Netherlands*, May, 2008.

[17] M. Kleiner, B. Dalenbäck, and P. Svensson, "Auralisation - an overview," *Journal of the Audio Engineering Society*, vol. 41, no. 11, pp. 861–875, 1993.

[18] L. Savioja, J. Huopaniemi, T. Lokki, and R. Väänänen, "Virtual environment simulation - advances in the diva project," in *International Conference on Auditory Display (ICAD'97), Palo Alto CA, USA*, 1997, pp. 43–46.

[19] B. Champagne, A. Lobo, and P. Kabal, "Efficient methods for simulating a moving talker in a rectangular room," in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, New Paltz, NY, USA*, 1991.

[20] B. Dalenbäck and M. Strömberg, "Real-time walkthrough auralization - The first year," in *Institute of Acoustics*, 2006.

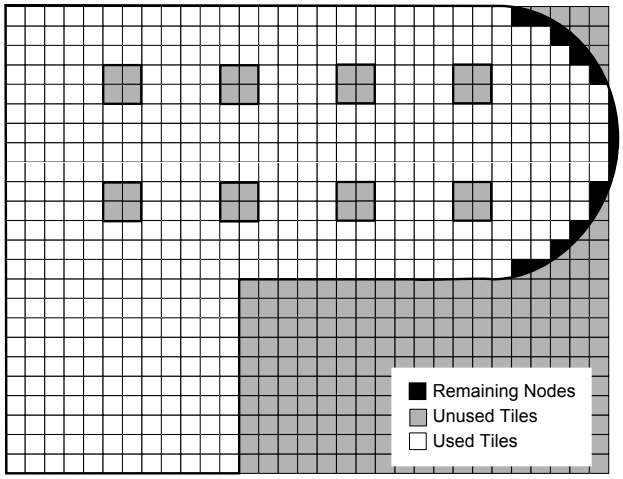

Fig. 10: An example of a complex geometry divided into tiles. Some areas cannot be divided equally into tiles the nodes in these areas are therefore remaining nodes and are shaded in black.

[21] R. Stewart and M. Sandler, "Generating a spatial average reverberation tail across multiple impulse responses," in *35th International Audio Engineering Society Conference, London, UK*, 2009.

[22] C. Masterson, G. Kearney, and F. Boland, "Acoustic impulse response interpolation for multichannel systems using dynamic time warping," in *35th International Audio Engineering Society Conference, London, UK*, 2009.

[23] A. Southern, J. Wells, and D. Murphy, "Rendering walkthrough auralisation using wave-based acoustic models," in *Proc. of 17th European Signal Processing Conference, EUSIPCO2009, Glasgow, UK*, August, 2009.

[24] P. Micikevicius, "3d finite difference computation on gpus using cuda," in *GPGPU2*, 2009.

[25] R. Abdelkhalek, H. Calandra, O. Couland, G. Latu, and J. Roman, "Fast seismic modelling and reverse time migration on a gpu cluster," in *2009 High Performance Computing and Simulation, HPCS'09*, 2009.

[26] S. Krakiwsky, L. Turner, and M. Okoniewski, "Graphics processor unit (gpu) acceleration of finite-difference time-domain (fdtd) algorithm," in *Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on*, vol. 5, 2004, pp. 265 –268.

[27] ——, "Acceleration of finite-difference time-domain (FDTD) using graphics processor units (GPU)," in *Microwave Symposium Digest, 2004 IEEE MTT-S International*, vol. 2, 2004, pp. 1033 – 1036.

[28] S. Adams, J. Payne, and R. Boppana, "Finite difference time domain (fdtd) simulations using graphics processors," in *DoD High Performance Computing Modernization Program Users Group Conference*, June 2007, pp. 334 –338.

[29] A. Balevic, L. Rockstroh, A. Tausendfreund, S. Patzelt, G. Goch, and S. Simon, "Accelerating simulations of light scattering based on finite-difference time-domain method with general purpose gpus," in *Computational Science and Engineering, 2008. CSE '08. 11th IEEE International Conference on*, July 2008, pp. 327 –334.

[30] A. Balevic, L. Rockstroh, W. Li, J. Hillebrand, S. Simon, A. Tausendfreund, S. Patzelt, and G. Goch, "Acceleration of a finite-difference method with general purpose gpus - lesson learned," in *Computer and Information Technology, 2008. CIT 2008. 8th IEEE International Conference on*, July 2008, pp. 291 –294.

[31] N. Takada, T. Shimobaba, N. Masuda, and T. Ito, "High-speed fdtd simulation algorithm for gpu with compute unified device architecture," in *Antennas and Propagation Society International Symposium, 2009. APSURSI '09. IEEE*, June 2009, pp. 1 –4.

[32] N. Röber, M. Spindler, and M. Masuch, "Waveguide-based room acoustics through graphics hardware," in *Proc. of ICMC06, New Orleans, USA*, November 2006.

[33] H. D. Campos, G., "On the computation time of three-dimensional digital waveguide mesh acoustic models," in *Euromicro Conference, 2000. Proceedings of the 26th*, 2000, pp. 332 – 339.

[34] N. Corporation, "Cuda programming guide 2.3," Online http://developer.download.nvidia.com/, 2009.

[35] N. Röber, U. Kaminski, and M. Masuch, "Ray acoustics using computer graphics technology," in *Proc. of the 10th Int. Conf. on Digital Audio Effects (DAFx-07),Bordeaux,France*, September 2007.