REAL-TIME 3D FINITE-DIFFERENCE TIME-DOMAIN SIMULATION OF LOW- AND MID-FREQUENCY ROOM ACOUSTICS

*Lauri Savioja**

NVIDIA Research,
Helsinki, Finland

Department of Media Technology,
Aalto University School of Science and Technology
Espoo, Finland
Lauri.Savioja@tkk.fi

ABSTRACT

Modern graphics processing units (GPUs) are massively parallel computing environments. They make it possible to run certain tasks orders of magnitude faster than what is possible with a central processing unit (CPU). One such case is simulation of room acoustics with wave-based modeling techniques. In this paper we show that it is possible to run room acoustic simulations with a finite-difference time-domain model in real-time for a modest-size geometry up to 7kHz sampling rate. For a 10% maximum dispersion error limit this means that our system can be used for real-time auralization up to 1.5kHz. In addition, the system is able to handle several simultaneous sound sources and a moving listener with no additional cost. The results of this study include performance comparison of different schemes showing that the interpolated wideband scheme is able to handle in real-time 1.4 times the bandwidth of the standard rectilinear scheme with the same maximum dispersion error.

1. INTRODUCTION

The graphics processing units (GPUs) of personal computers have developed a lot during the last decade and nowadays they are actually massively parallel computing environments that have plenty of applications outside graphics. In practice, GPUs have enabled running of certain high-performance computing tasks on everyones desktop. Especially, in room acoustics, employing a GPU in wave-based simulations has opened completely new possibilities, even so that real-time simulation on a limited bandwidth is possible.

The finite-difference time-domain (FDTD) technique is a wave-based method that is particularly suitable for GPU computation. FDTD simulations are commonly used in solving various fluid-dynamic problems although in room acoustics they have not been that popular. One reason for this is that FDTD simulations are remarkably computation intensive. Especially at higher frequencies the computational load can be excessive. For this reason, the wave-based methods are typically used only at the lowest frequencies. However, that is the range where the wave nature of sound is dominant, and all the other modeling approaches, such as the geometrical acoustics, fail.

In this paper we present a novel system capable of auralizing arbitrary 3D geometries with digital impedance filter boundaries in real-time. In this context the term *auralization* means both modeling of sound propagation and making those modeling results audible. Our system is able to handle spaces of 100 m^3 up to 7kHz sampling rate. The system uses the central processing unit (CPU) to handle audio input and output, to perform sample rate conversions and required filters whereas the actual acoustic FDTD simulation is run on the GPU. By using this system, we compare the computational performance of different explicit FDTD schemes in practice. The results show the expected superiority of the interpolated wideband scheme. Another investigated case is the computational load caused by different boundary conditions. The main conclusion is that computation of the frequency indepedent boundaries is much faster than the use of digital impedance filters.

This paper is organized as follows. Section 2 covers the relevant literature regarding GPU computation and room acoustic modeling. Section 3 presents the applied FDTD method while Section 4 discusses the basics of GPU programming. Our implementation and results are discussed in Sections 5 and 6, respectively. Finally, Section 7 concludes the paper.

2. BACKGROUND

2.1. Room acoustic modeling

There are basically two different approaches for room acoustic modeling, the wave-based methods and the ray-based methods [1]. The former target at solving the wave equation numerically whereas the latter neglect all the wave phenomena and sound is supposed to act like rays. Both of them have advantages: the wave-based methods excel at low frequencies while the ray-based methods are most suitable for high frequencies.

Ray-based modeling

The ray-based approach, also known as the geometrical acoustics (GA), has been used for over 40 years as the basic principle of acoustic ray tracing was presented in 1968 by Krokstad et al. [2]. The other main GA technique, the image source method, was introduced by Allen and Berkley in 1979 [3]. Those two form the base for most of the room acoustic design software currently in use. Both of these techniques have been developed further, and numerous new variants of them have emerged. Lately, a unifying mathematical framework for all the ray-based techniques, the room acoustic rendering equation, was presented by Siltanen et al. [4].

* This work was supported by the Academy of Finland (Project #130062).

Wave-based modeling

The currently used wave-based modeling methods can be roughly categorized into three classes: finite element (FE), boundary element (BE), and finite-difference time-domain (FDTD) methods [1]. In both FE and FDTD methods the space under study is discretized into volumetric elements while in the BEM the boundaries of the space are discretized. Their computational loads depend directly on the number of elements in the mesh such that the smaller the element, the more elements that are needed and the wider the covered bandwidth is. Typically these methods are employed only for modeling of low frequency behavior of rooms. The FE and BE methods operate typically in the frequency-domain and are best suited for modeling of static scenes. Out of these methods, the FDTD technique is the most suitable for real-time auralization. With a precomputation step even the BEM results can be used for auralization of static scenes.

In FDTD methods the simulation progresses iteratively in time steps. In the traditional FDTD methods two variables, sound pressure and volume velocity, are needed and their values are updated alternately [5]. However, it is possible to use only one variable as well. The FDTD schemes can be divided into two groups: the explicit schemes and the implicit schemes. The explicit schemes are straightforward to implement as the values of the next time step are computed solely based on the previous time steps. The implicit FDTD schemes are more complicated as the new value of a node depends on the new values of neighboring nodes thus requiring simultaneous solving of all nodes. In this paper, we limit ourselves to one-variable explicit schemes although there exists intriguing implicit schemes, such as the alternating direction implicit (ADI) method [6], as well.

In all FDTD methods numeric dispersion is an inherent problem, and it affects the bandwidth which can be considered valid. Kowalczyk and van Walstijn have investigated various FDTD schemes thoroughly and according to their results the interpolated wideband scheme (IWB) is the most efficient one from this point of view [7]. The other scheme used in this study is the standard rectilinear (SRL) scheme. It is worth noting that the SRL scheme is the same as the original digital waveguide mesh [8, 9], and it has been called also as the standard leapfrog scheme [7]. In this study, we are interested only in the maximum dispersion, and thus methods removing the direction dependency in the dispersion, such as in the interpolated digital waveguide mesh [10], are not relevant here. In addition to dispersion, another challenge in FDTD modeling has been the incorporation of frequency dependent boundary conditions, but there is a recent well-described solution, called digital impedance filters, to that problem as well [7].

A competing approach to the previously listed ones, called Adaptive Rectangular Decomposition (ARD), has been introduced by Raghuvanshi et al. [11]. It is efficient and suits well for GPU implementation. The fundamental difference of ARD when compared to the other wave-based methods is that the space is divided into maximally large rectangular subspaces, and inside each such subspace no further discretization is needed, and the main computation lies in their interconnection.

Real-time auralization

The geometrical acoustics (GA) has been the base for real-time auralization systems presented so far such as in the DIVA system [12] and in the IKA-sim [13]. In those systems the sound propagation modeling and audio signal processing are separated from each other such that there is a GA simulation engine computing reflection paths. It provides them to the signal processing module that makes the paths audible. However, in practice the amount of reflection paths gets really large, and only a fraction of them can be handled individually and the rest of the paths have to be treated by statistical means. This means that a diffuse reverberation unit is needed and it will take care of the late part of the room impulse response. Although those models may sound convincing, they lack the physical base in the low frequency region as there is e.g. no higher-order diffraction modeling and the phase of sound waves is typically neglected. Incorporating those effects is difficult or even impossible in the ray-based models. At best there has been systems capable of modeling first order edge diffraction [14, 15], but they can be considered highly approximative when compared to the accuracy provided by the wave-base methods. One recent technique, the frequency domain acoustic radiance transfer by Siltanen et al. [16] presents another view on implementing real-time auralization. It needs heavy pre-computation, but can cover the whole impulse response in full bandwidth such that there is no need for a separate reverberation algorithm. However, the technique is suitable only for static geometries and stable sound sources and there is no diffraction modeling. As far as we know there hasn't been any implementation of any wave-based technique capable of real-time auralization.

The ultimate target in room acoustic simulation would be to handle the whole audible frequency range with the more accurate wave-based methods, but that goal is not reachable in the near future as in those methods the workload grows steeply as a function of the sampling frequency. In the techniques based on volumetric grids, such as FDTD and FE methods, doubling the upper frequency limit doubles the grid density in all three dimensions and in addition the number of time steps gets doubled, altogether the increase in the computational load is $2^4 = 16$ fold. So, if we today reach 4.4kHz sampling frequency with an FDTD simulation we need more than 10^4 times as much computation to go up to 44.1kHz with the same method. Thus the best option still in the near future is to have a hybrid method in which different frequency ranges are modeled with different techniques as suggested e.g. by Savioja et al. [17] and Murphy et al. [18].

2.2. GPU compute

General purpose GPU (GPGPU) computation has been under active investigation since the introduction of programmable shaders in 2001 [19]. Even after that the GPUs have been targeted mainly for graphics, but the trend has been to increase the programmability and to make the use of GPUs for non-graphics tasks as flexible as possible. The old fixed-function graphics pipeline doesn't dictate the GPU computing capabilities anymore. At the moment a programmer can see the GPU just as a massive collection of parallel computing units, and the programmer doesn't have to know anything of graphics programming.

The early days of GPGPU computation concentrated mostly on getting around the limitations set by the graphics programming APIs (application programming interface) although there are some remarkable results already from that time as presented by Owens et al. [20]. Only after introduction of the CUDA (compute unified device architecture) API by NVIDIA in 2006 [21] the GPU computation has become really popular. At the moment, GPUs are commonly used in computational sciences including such areas as electromagnetics, astrophysics and brain research [22].

In DSP (digital signal processing) GPUs have been proven to be very effective in certain operations such as in fast Fourier tranforms [23], or in additive synthesis [24]. Use of GPUs on audio related tasks has been studied earlier e.g. by Tsingos [25].

GPU-enhanced room acoustic modeling

Ray-based room acoustic modeling shares several similarities with global illumination computation in computer graphics. Thus there is a lot of computer graphics literature that directly benefits room acoustic modeling. The first acoustic ray-tracing study utilizing GPUs was published already in 2004 [26]. Another GPU-based acoustic ray-tracing system with auralization capabilities has been presented by Röber et al. [27]. The frequency domain acoustic radiance tranfer is another example, in which the signal processing required for auralization is performed on GPU [16]. In addition, there are algorithms that have been developed to be more generally parallelizable such that they are suitable for any multi-core processor, see e.g. [28, 29].

In this paper we concentrate on the frequency range where wave phenomena are essential, and thus our focus is on the wave-based methods. Most of them can be parallelized and will benefit from use of GPUs. First studies in which GPUs were utilized to solve partial differential equations with FEM have been published by Rumpf and Strzodka [30]. The recent adaptive rectangular decomposition technique is another example of wave-based modeling in which GPUs offer a remarkable performance gain [11].

The GPUs can help with the boundary element methods as well and it is even possible to solve acoustic problems of realistic size [31]. One simplified version of the boundary element method is called the Kirchoff approximation. It can be used to compute first order diffractions very efficiently. It has been shown that using a GPU, it is possible to compute sound scattering at interactive rates for up to 20 point frequencies in a model containing almost 100,000 triangles [32].

However, it seems that out of the wave-based techniques the FDTD method is the most straightforward to parallelize. In an FDTD simulation, the computation can be distributed to several processors operating independently from each other. Röber et al. were the first to report GPU implementation of the digital waveguide mesh technique [33]. They achieved almost 70 fold performance gain over a CPU implementation in a 2D case. They also show results with 3D meshes, but they are not that impressive due to the limitations of display drivers of that time. Another study investigating parallelization of the digital waveguide mesh method has been presented by Campos et al. in 2000 [34]. They used a 8-processor high-end SGI computer to test the performance gain achievable by parallelization, and in the best case they report linear gain such that doubling the number of processors doubled the performance as well. A very recent study on 2D acoustic modeling with GPU accelerated FDTD implementation reports over 70 fold performance increase over a CPU implementation with a one-million node mesh [35].

3. FINITE-DIFFERENCE TIME-DOMAIN SIMULATION USING COMPACT EXPLICIT SCHEMES

Typically the FDTD methods use rectangular grids although there are numerous other possible mesh topologies as well. In this study, we are limited to rectangular grids and compact schemes meaning that in the computation only the nearest neighbors, in axial, 2D diagonal, and 3D diagonal directions, of a node are needed in the computation of its new value. Thus we need to discretize the space to be modeled into a regular grid in which each interior node has a connection to its 26 neighbors and one to itself. Kowalczyk et al. have analyzed this scheme family and we utilize the coefficients they suggest [7]. According to their notation the mesh is governed by the following equation:

$$\begin{aligned} p^{n+1}_{l,m,i} = {} & d_1(p^n_{l+1,m,i} + p^n_{l-1,m,i} + p^n_{l,m+1,i} + \\ & \quad p^n_{l,m-1,i} + p^n_{l,m,i+1} + p^n_{l,m,i-1}) \\ & + d_2(p^n_{l+1,m+1,i} + p^n_{l-1,m+1,i} + p^n_{l-1,m-1,i} + \\ & \quad p^n_{l+1,m-1,i} + p^n_{l+1,m,i+1} + p^n_{l+1,m,i-1} + \\ & \quad p^n_{l-1,m,i-1} + p^n_{l-1,m,i+1} + p^n_{l,m+1,i+1} + \\ & \quad p^n_{l,m+1,i-1} + p^n_{l,m-1,i-1} + p^n_{l,m-1,i+1}) \\ & + d_3(p^n_{l+1,m+1,i+1} + p^n_{l-1,m+1,i+1} + p^n_{l+1,m-1,i+1} + \\ & \quad p^n_{l-1,m-1,i+1} + p^n_{l+1,m+1,i-1} + p^n_{l-1,m+1,i-1} + \\ & \quad p^n_{l+1,m-1,i-1} + p^n_{l-1,m-1,i-1}) \\ & + d_4 p^n_{l,m,i} - p^{n-1}_{l,m,i} \end{aligned} \tag{1}$$

with the coefficients

$$d_1 = \lambda^2(1 - 4a + 4b), \qquad d_2 = \lambda^2(2 - 2b),$$
$$d_3 = \lambda^2 b, \qquad d_4 = 2(1 - 3\lambda^2 + 6\lambda^2 a - 4b\lambda^2).$$

where $p^n_{l,m,i}$ stands for the sound pressure at time step n in location $[l, m, i]$. The values of the coefficients λ, a and b are determined by the chosen FDTD scheme. It is worth noting that the sampling rate of the mesh depends on λ such that $f_s = \frac{c}{\lambda \Delta x}$, where Δx is the grid spacing and c represents the speed of sound. The digital waveguide mesh methods form a subset of these schemes in which the relation of wave propagation speed and Δx is fixed based on the mesh topology, for example with a rectangular 3D digital waveguide mesh the $\lambda = \sqrt{\frac{1}{3}}$.

Scheme	a	b	λ	d_1	d_2	d_3	d_4
SRL	0	0	$\sqrt{\frac{1}{3}}$	$\frac{1}{3}$	0	0	0
IWB	$\frac{1}{4}$	$\frac{1}{16}$	1	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{5}{8}$	$-\frac{3}{2}$

Table 1: The key parameters of the standard rectilinear (SRL) and the interpolated wideband (IWB) schemes.

In this study we concentrate on two of the schemes, the standard rectilinear (SRL) and the interpolated wideband (IWB) scheme. Their key parameters are listed in Table 1. The SRL scheme was chosen since it is computationally efficient as only one of the d_i coefficients, d_1, is non-zero such that only six neighbors are involved in the computation of Eq. (1). The other sparse schemes having a zero value for some of the coefficients d_i are not that efficient and thus they were left out. The IWB scheme was selected since it has been shown to cover the widest frequency range up to $0.5 f_s$ still having least dispersion [7]. This means that out of the full schemes with all d_i coefficients having a non-zero value, the IWB scheme suits best for real-time auralization.

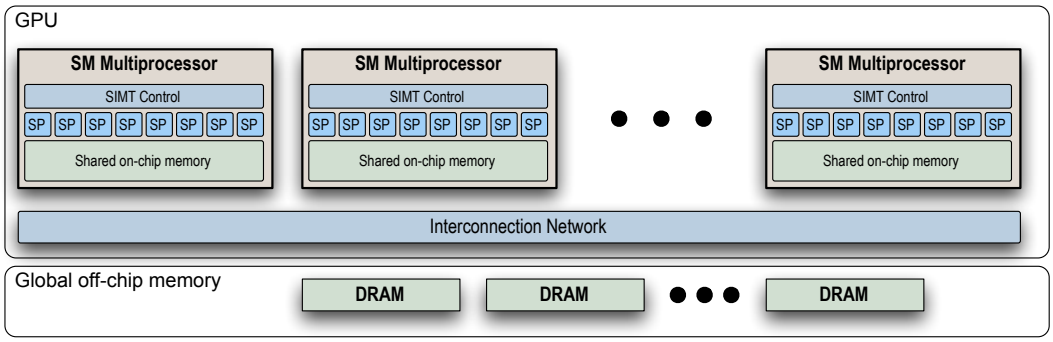

Figure 1: Typical GPU consists of dozens of streaming multiprocessors (SM) each having several parallel scalar processors (SP) operating in single-instruction multiple-thread (SIMT) manner. All the SMs share a large off-chip global memory.

Boundary and edge nodes

Kowalczyk et al. describe how to implement the boundaries with digital impedance filters (DIF) [36, 7]. In the derivation of that boundary model there are so called ghost points that lie outside of the actual mesh. In the final equations those nodes are eliminated such that each ghost point is replaced by an infinite impulse response (IIR) filter. A zeroth-order filter corresponds to frequency independent impedance boundary condition. With higher filter orders it is possible to describe frequency dependent boundary conditions. It is worth noting that implementation of frequency independent boundaries is clearly less memory consuming than that of higher-order DIFs since there is no need for the actual DIF filter. To find the equations for various boundary node types, see [7].

4. PROGRAMMING A GPU

Typical structure of a modern GPU is illustrated in Fig. 1. In this structure the GPU contains several streaming multiprocessors (SM) that are connected to each other by a interconnection network. Furthermore, one SM consists of parallel processors that share a fast access on-chip memory whereas data exchange between different SMs needs use of relatively slow global memory. Inside one SM, each processor executes the same instructions whereas different SMs can have different execution patterns.

There are several possible APIs to program GPUs, and there is no de-facto standard, yet. The OpenCL by Khronos Group [37] is an attempt to that direction, but it is too early to predict if it will attain enough of popularity to became the dominant API. At the moment, the most popular API is the CUDA API by NVIDIA [21]. Although it is vendor-specific, we have used it in this study as it is the most mature API and quite similar to OpenCL.

The computation model in CUDA is based on threads that are run in parallel on the GPU. CUDA provides a SIMT (single instruction multiple thread) interface to the GPU. In that model the programmer writes a *kernel* that describes the behavior of one thread, and after that the programmer has to launch enough of threads to accomplish the task. The underlying CUDA runtime takes care of actual running of those threads in parallel. This paradigm is especially suitable for data-parallel problems, in which the task can be described from point-of-view of one data item. This is the case for FDTD simulations, since it is sufficient to have a kernel that computes the value of Eq. (1) in one node, and then launch one thread for each node in the mesh.

In CUDA, threads are grouped into *warps*. Each warp is executed in one SM such that all the threads in a warp have the same execution pattern. This means that if there is even one thread needing some extra computation due to branching, all the threads in that warp have to wait for the completion of divergent branch execution of that thread. For this reason, it would be beneficial to group the threads such that there is no branching inside a warp. Different warps can have different execution patterns without any extra performance penalty.

Although the processors on a GPU have enormous computation capability in total, it is not obvious how to fully utilize it in practice. It is easy in cases where the amount of data is relatively small, but in more typical cases the memory bandwidth between the global memory and SMs becomes a bottleneck. For example, a FDTD simulation with one million nodes and update rate of 44.1kHz would need a data rate of at least 500GB/s (3 layers/update $\times$ 10^6 nodes/layer $\times$ 44100 updates/second $\times$ 4 bytes/node) and thus make it difficult to implement as the current memory bus bandwidths are around 80-100 GB/s.

In addition to the bandwidth limit, there is a remarkable latency in fetching data from the global memory. The on-chip memory, called the shared memory, is much faster than the global memory. However, use of the shared memory is more complicated and needs special attention. In this study the shared memory is not utilized at all. Part of the on-chip memory is dedicated to store constants, and is called the constant memory. That memory area is writable only from the CPU code, and the GPU threads have only read-only access to that memory.

To hide the memory latencies, it is desirable to have as many threads as possible in execution at a time. It is essential that there are always some threads that can be executed while the others wait for their memory fetches to finish. In practice, it is preferable that there are at least tens of thousands of threads running in parallel.

Another technique to reduce the performance penalty caused by the limited memory bandwidth and latency, is the use of cache memory that provides fast access to the most often needed data items. This concept has been used in CPUs for a long time but has been practically missing in GPUs. Fortunately, this is not the case anymore as e.g. in the Fermi architecture by NVIDIA [38] there is a two-level cache hierarchy. It is a major improvement as it makes programmers' work much easier regarding performance optimization. However, it is still important to write the code so that there is a high level of cache utilization in order to realize the

maximum performance benefits.

5. IMPLEMENTATION

The target of our implementation was to enable real-time auralization for a limited bandwidth. The design specification included sound propagation modeling in arbitrary 3D geometries from multiple simultaneous sound sources, and play back of the modeling results for a moving listener. The resulting system consists of several parts, and it employs both the GPU and the CPU.

Overall, the system (1) reads audio streams from files, (2) downsamples them for the FDTD simulation, (3) feeds the signals to the sound source positions in the mesh, (4) computes the mesh, (5) reads the output signal from the listener position, (6) upsamples that signal, (7) sends it to the audio output device. At the moment the audio output is only monophonic. In the implementation, we wanted to allocate the GPU completely to the FDTD computation, and everything else is handled by the CPU.

5.1. Signal processing at CPU

The CPU was used to perform the required sampling rate conversions. To keep this part simple, we restricted the possible conversions only to integer factors. For example, for a 48kHz input signal the downsampling factor can be e.g. 8 or 10 corresponding to the mesh update rates of 6kHz and 4.8kHz, respectively. The applied anti-aliasing filters were Kaiser-windowed sinc-functions [39]. Although this is not the optimal solution regarding the filter lengths, it was clearly sufficient for this purpose since the mesh computation on the GPU was the main bottleneck in computation. The applied anti-alias filters were 256 tap long to guarantee high quality conversions.

5.2. FDTD modeling on GPU

Computation of one time step in the FDTD simulation consists of two consecutive kernel launches. In the first launch there are N threads where N is the number of nodes in the mesh. It takes care of updating the actual mesh for both nodes inside the mesh and on the boundaries. In addition, it handles the nodes with a sound source or a listener. In the second launch the number of threads equals the number of boundary filters, and that is used to update the DIFs (see Eqs. (39)-(41) in [7]).

Although the actual FDTD simulation is done one time step at a time on the GPU, the communication between CPU and GPU is done in larger blocks such that the input and output signals are copied to and from the GPU memory in blocks. According to our experiments it seems that increasing the block size above 64 samples doesn't increase the performance any more.

Updating the actual FDTD mesh

Computation of one time step according to Eq. (1) requires information from two previous time steps. However, it is possible to store the values of time step $n+1$ on the same memory locations as the values of time step $n-1$ such that it is sufficient to have only two separate memory areas, instead of three, to handle the mesh. In addition, we store the node type explicitly for each node. It tells if the node is a sound source or a listener. For boundary nodes we explicitly store either the impedance of the boundary or an identifier referring to the DIF used by that node. The memory needed for the mesh, node types and DIFs is allocated from the global memory. To speed up the computation of the mesh we use the texture cache of the GPU. That is a one-level cache that has to be configured by hand. In the future GPUs, such as in the Fermi architecture by NVIDIA, there is no need to use this texture cache anymore since there is real two-level cache hierarchy. All the constants needed in the computation, such as d_i and λ, are stored in constant memory.

The kernel applied in node updates is quite simple. It first fetches the values of all the neighbors at time step n and the value of the node itself at time steps n and $n-1$. After that the type of the node is checked. Most of the nodes are inside the mesh, and for them the code corresponding Eq. 1) is called. For boundary and edge nodes we use their corresponding equations. In the end, the new computed value of the node is stored to replace the value of time step $n-1$. It is worth noting, that the only divergent branching in the kernel is for the actual value computation as all the memory fetches are performed for all the node types. In addition, we have separate kernels for both schemes such that the SRL computation can truly benefit of having to fetch only 6 neighbor values instead of 27 values needed in updating with the IWB scheme.

Sound sources and listeners are treated by the same kernel but their speciality is that they have their own buffers for input and output signals. If a node is a sound source, there is a plain assignment statement as the new excitation value is read from the input buffer and set to the value of the node. Listeners are fully transparent, and they are updated similarly to other mesh interior nodes. In addition to that, the value of time step n is stored to the listener output buffer. Actually, there are always two listener nodes for one listener to enable smooth movements in the scene. With this technique we can compute the actual output signal as a linear cross-fade of those two listener signals thus avoiding transients when a listener moves from one node to another.

Updating boundary filters

In the construction of the mesh, we allocate one DIF for each boundary node with a filter of order one or larger. The DIF update kernel is such that one kernel will handle one DIF. The computation time depends on the order of the filter as is the case for any IIR filter. The coefficients of the impedance filters are precomputed and they are stored in the constant memory. The memory needed for the actual filters is allocated from the global memory. For performance reasons it is advantageous to organize the memory such that threads in the same warp will access memory locations close to each other.

6. SIMULATION RESULTS

The target of this study was to find out what frequency range can be simulated in real-time for spaces of different size.

6.1. Simulation setup

Geometries

The performance of FDTD computation was tested in two different rooms, the first one sized of a typical living room, and the second one had a volume of a concert hall as listed in Table 2. For testing the performance of the DIFs we used a third geometry. Plain shoebox shaped rooms are too simplistic to be generalizable for

Space	Length	Width	Height	Volume
Room	$7m$	$5m$	$2.8m$	$98m^3$
Hall	$40m$	$20m$	$15m$	$12000m^3$

Table 2: The dimensions of the two spaces used in the study.

realistic use scenarios. For this reason, we added more surfaces to the concert hall such that the third setup had the same volume as the concert hall, but the space was divided into 12 smaller rooms thus increasing the number of boundary nodes.

Computer setup

The tests were run on a commodity PC having Intel Pentium Dual CPU E2180 running at 2.00GHz and 2.0GB of RAM memory. The GPU was Nvidia Quadro FX 5800 with 4.0GB of RAM memory. The GPU was connected to the PCIe (PCI Express) bus of the mother board having 2 GB/s bandwidth.

The system was compiled and run with Microsoft Studio 2005 development environment under the Microsoft Windows XP Pro SP3 operating system. For audio playback the Windows WaveOut API was used. The GPU code used the NVIDIA CUDA library [21].

6.2. Mesh computation performance

Test procedure

The performance of a simulation was measured by running it for 512 steps and measuring the time taken by that. For each case, the maximum update frequency f_s was searched by gradually decreasing the Δx. As long as the measured time was shorter than the actual duration of those samples, the simulation was considered real-time. It is worth noting that the chosen schemes have different values of λ, and thus the same values of f_s correspond to different values of Δx. In this setup, the boundary nodes were set to have a frequency-independent impedance.

The chosen schemes can not be compared just by looking at the maximum f_s since they have different dispersion characteristics. For this reason we set a threshold for the maximum allowed dispersion, and that is used to get the upper limit frequency f_l describing the actual valid bandwidth. There is no generally defined maximum dispersion error for auralization, and the audibility of dispersion depends heavily e.g. on the distance from the sound source. In this study we use quite loose limit of 10% error that still should be usable for some cases. The data presented by Kowalczyk and van Walstijn [7] was used to determine this 10% dispersion limit for both schemes, resulting in the following values, for the SRL $f_l = 0.16 f_s$, and for the IWB $f_l = 0.37 f_s$. The maximum dispersion error curves for both schemes are roughly of the same shape and thus their relative performance does not depend on this error limit but affects only the absolute valid bandwidth.

Results

The obtained results are presented in Table 3, in which N corresponds to the number of nodes in the mesh, and the last column shows the relative performance compared to the SRL scheme in the same setup. The maximum f_l for the room geometry was 1.5kHz, and thus it well covers the frequency range in which wave-based modeling is essential. Even for the concert hall-sized geometry the acoustics can be simulated up to almost 500Hz.

Space	Scheme	Δx (cm)	N	f_s (Hz)	f_l (Hz)	rel.
Room	SRL	8.66	150336	6857	1063	1.00
Room	IWB	8.46	161601	4056	1509	1.42
Hall	SRL	27.13	598290	2190	339	1.00
Hall	IWB	27.21	590205	1260	469	1.38

Table 3: Simulation performance of the SRL and IWB schemes in two different geometries. Although SRL has a higher update rate f_s, the valid frequency range f_l of IWB is 1.4 times that of the SRL.

The results show clearly that the IWB scheme provides roughly 1.4 times the bandwidth of the SRL scheme meaning that IWB is about $1.4^4 \approx 3.8$ times more efficient than SRL. It seems that the resulting mesh sizes (N) are almost the same for both schemes although their sampling rates have a remarkable difference. This difference is caused by the fact that in the SRL scheme only six neighbors are involved in the computation whereas in the IWB scheme all 27 possible nodes are needed. This causes much more computation and many more memory fetches, although most of them should hit in cache.

Another result we obtained was that the number of sound sources does not affect the performance in practice. We ran the same setups with both one and eight simultaneous sound sources and the results remained the same.

6.3. Digital impedance filter computation performance

Test procedure

The digital impedance filter performance was tested in two geometries, in the hall and in the hall which was divided into 12 smaller volumes. We used the same discretization of $\Delta x = 28cm$ in all the cases. This resulted in total amount of 505 908 nodes out of which 7.5% were boundary nodes in the hall meaning that there were 41 298 DIFs to be computed. In the 12 room hall the boundary node ratio was 14.1%. As a reference result we made a simulation with a mesh with the same amount of nodes but such that all of them were treated as normal nodes inside the mesh and there were no boundary nodes at all. The simulation was run for different filter orders up to 10th order such that filter order 0 corresponded to a node with frequency independent impedance and they were treated without actual DIF computation. For each case, the computation time of 512 time steps was recorded.

Results

Results of the DIF performance test are illustrated in Fig. 2. The curves show the relative increase in the computation time as a function of the filter order. The results show that additional computational cost of the plain frequency independent impedance, i.e. order 0, is minimal, even in the worst case it is less than 12%. Instead, introduction of the DIFs increase the computation time remarkably. However, increasing the filter order above one increases the computation time only modestly. The two different geometries behave as expected such that in the hall model the relative increase is smaller since the proportion of the boundary nodes is smaller

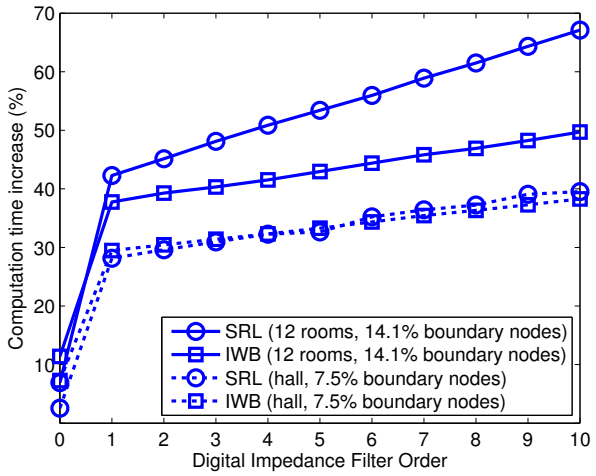

Figure 2: The relative increase in computation time of the whole mesh as a function of the filter order such that in case '0' the boundaries have a frequency independent impedance and there is no actual filter. Need for the filter clearly increases the computational load. The reference in all the cases is a mesh of the same size but without any boundary nodes.

than in the 12 room case. Similarly, the relative cost of DIF boundaries with the SRL scheme is larger since the filter update costs are equal in both schemes but the cost of the actual node update is smaller in the SRL scheme.

Overall, the relative cost of DIFs is quite large. This means that they should be used only in the case where they are really needed if the target is to have real-time auralization. For non-real-time use, the situation is completely different and there is no reason to avoid DIFs. However, it seems that if DIFs are to be used, the filter order can be chosen quite freely. The obtained results raise an interesting question of the optimal strategy to handle boundaries. If it is acceptable to have frequency independent impedance inside an octave, then there could be a separate mesh for each octave band and bandpass filters to feed each octave band to corresponding mesh. The reasoning behind this is the fact that the computational load of a mesh an octave lower is only $1/16^{th}$, i.e. 6.25%, of that of the original mesh. However, the ratio of boundary nodes increases and discretization errors come larger when the grid spacing increases. Thus it is not obvious which strategy to implement the boundaries is the best one.

6.4. Future work

This study opens up several possibilities for future research. In our opinion, the most crucial one is perceptual evaluation of the FDTD simulation itself. The dispersion error affects severely the quality of simulations and with this tool it is possible to interactively search for the limit of dispersion audibility in different geometries. Another topic that deserves perceptual evaluation lies in the area of boundary modeling as mentioned in Section 6.3.

7. CONCLUSIONS

The target of this study was to find a way to make real-time wave-based simulation of room acoustics possible with the help of parallel computation capabilities of a modern GPU. That target was achieved, and the results show that a space of ca. $100m^3$ can be simulated in real-time such that the results are within 10% dispersion error limit up to 1.5kHz.

Now, for the first time it is possible to have a real physically-based real-time room acoustic simulation in the frequency range where modeling the wave phenomena is essential. In addition to being an auralization tool as such, this work makes further studying of the perceptual quality of FDTD simulations much easier. Especially, audibility of the dispersion error and the accuracy needed in boundary modeling should be investigated more.

8. ACKNOWLEDGEMENTS

PhD K. Kowalczyk and PhD M. van Walstijn are acknowledged for commenting the manuscript and for helpful discussions regarding the interpolated wideband scheme and the digital impedance filters. PhD T. Aila, Mr A. Southern, PhD T. Lokki, and PhD D. Murphy are thanked for commenting the manuscript. Mr T. Karras, PhD T. Aila, and PhD S. Laine from NVIDIA Research are thanked for their practical help with CUDA and GPU programming.

9. REFERENCES

[1] U. P. Svensson and U. Kristiansen, "Computational modelling and simulation of acoustic spaces," in *Proc. AES 22nd Conf. on Virtual, Synthetic and Entertainment Audio*, pp. 11–30. Espoo, Finland, June 2002.

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

[3] J. Allen and D. Berkley, "Image method for efficiently simulating small-room acoustics," *Journal of the Acoustical Society of America*, vol. 65, no. 4, pp. 943–950, 1979.

[4] S. Siltanen, T. Lokki, S. Kiminki, and L. Savioja, "The room acoustic rendering equation," *Journal of the Acoustical Society of America*, vol. 122, no. 3, pp. 1624–1635, 2007.

[5] D. Botteldooren, "Finite difference time domain simulation of low frequency room acoustic problems," *The Journal of the Acoustical Society of America*, vol. 98, no. 8, pp. 3302–3308, Jan 1995.

[6] M. van Walstijn and K. Kowalczyk, "On the numerical solution of the 2D wave equation with compact FDTD schemes," in *Proc. 11th Int. Conference on Digital Audio Effects (DAFx-08)*. Espoo, Finland, Sept. 2008.

[7] K. Kowalczyk and M. van Walstijn, "Room acoustics simulation using 3-D compact explicit FDTD schemes," *IEEE Transactions on Audio, Speech and Language Processing*, p. in print, 2010.

[8] S. Van Duyne and J. O. Smith III, "The tetrahedral digital waveguide mesh," in *Proc. IEEE ASSP Workshop on Applications of Signal Processing to Audio and Acoustics*. Mohonk, New Paltz, NY, Oct. 1995.

[9] L. Savioja, T. Rinne, and T. Takala, "Simulation of room acoustics with a 3-D finite difference mesh," in *Proc. Int. Computer Music Conf.*, pp. 463–466. Aarhus, Denmark, Sept. 1994.

[10] L. Savioja and V. Välimäki, "Interpolated rectangular 3-D digital waveguide mesh algorithms with frequency warping," *Speech and Audio Processing, IEEE Transactions on*, vol. 11, no. 6, pp. 783 – 790, Nov 2003.

[11] N. Raghuvanshi, R. Narain, and M. Lin, "Efficient and accurate sound propagation using adaptive rectangular decomposition," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 5, pp. 789 – 801, 2009.

[12] L. Savioja, J. Huopaniemi, T. Lokki, and R. Väänänen, "Creating interactive virtual acoustic environment," *Journal of the Audio Engineering Society*, vol. 47, no. 9, pp. 675–705, Sept. 1999.

[13] A. Silzle, H. Strauss, and P. Novo, "IKA-SIM : A system to generate auditory virtual environments," in *116th AES Convention*. Berlin, Germany, 2004.

[14] P. Calamia, U. P. Svensson, and T. Funkhouser, "Integration of edge-diffraction calculations and geometrical-acoustics modeling," in *Proceedings of Forum Acusticum*. Budapest, Hungary, Sept. 2005.

[15] M. Taylor, A. Chandak, Z. Ren, C. Lauterbach, and D. Manocha, "Fast edge-diffraction for sound propagation in complex virtual environments," in *In Proc. EAA Auralization Symposium*. Espoo, Finland, June 2009.

[16] S. Siltanen, T. Lokki, and L. Savioja, "Frequency domain acoustic radiance transfer for real-time auralization," *Acta Acustica united with Acustica*, vol. 95, pp. 106–117, 2009.

[17] L. Savioja, J. Backman, A. Järvinen, and T. Takala, "Waveguide mesh method for low-frequency simulation of room acoustics," in *Proc. 15th Intl. Congress on Acoustics (ICA)*. Trondheim, Norway, June 1995.

[18] D. Murphy, M. Beeson, S. Shelley, and A. Moore, "Hybrid room impulse response synthesis in digital waveguide mesh based room acoustics simulation," in *Proc. 11th Int. Conference on Digital Audio Effects (DAFx-08)*, pp. 1–8. Espoo, Finland, Sept. 2008.

[19] E. Lindholm, M. Kilgard, and H. Moreton, "A user-programmable vertex engine," in *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. Los Angeles, CA, USA, Aug. 2001.

[20] J. Owens, D. Luebke, N. Govindaraju, and M. Harris, "A survey of general-purpose computation on graphics hardware," *In Eurographics 2005, State of the Art Reports*, pp. 21–51, Sept. 2005.

[21] NVIDIA Corporation, "NVIDIA CUDA programming guide," pp. 1–147, 2009.

[22] NVIDIA Corporation, "GPU technology conference," 2009, http://www.nvidia.com/object/gpu_technology_conference, accessed April 6th, 2010.

[23] N. Govindaraju, B. Lloyd, Y. Dotsenko, B. Smith, and J. Manferdelli, "High performance discrete Fourier transforms on graphics processors," *In Proc. SC '08: ACM/IEEE conference on Supercomputing*, Nov. 2008.

[24] L. Savioja, V. Välimäki, and J. O. Smith III, "Real-time additive synthesis with one million sinusoids using a GPU," in *AES 128th Convention*. London, UK, 2010.

[25] N. Tsingos, "Using programmable graphics hardware for auralization," in *Proc. EAA Symposium on Auralization*. Espoo, Finland, June 2009.

[26] M. Jedrzejewski and K. Marasek, "Computation of room acoustics using programmable video hardware," in *Proc. International Conference on Computer Vision and Graphics, ICCVG 2004*, pp. 587–592. Warsaw, Poland, Sept. 2004.

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

[28] A. Chandak, C. Lauterbach, M. Taylor, Z. Ren, and D. Manocha, "AD-Frustum: Adaptive frustum tracing for interactive sound propagation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, pp. 1707 – 1722, 2008.

[29] M. Taylor, A. Chandak, L. Antani, and D. Manocha, "RESound: Interactive sound rendering for dynamic virtual environments," in *In Proc. 17th International ACM Conference on Multimedia*. Beijing, China, Oct. 2009.

[30] M. Rumpf and R. Strzodka, "Using graphics cards for quantized FEM computations," *In Proc. of VIIP*, pp. 193–202, 2001.

[31] T. Takahashi and T. Hamada, "GPU-accelerated boundary element method for Helmholtz'equation in three dimensions," *International Journal for Numerical Methods in Engineering*, vol. 80, no. 10, pp. 1295–1321, 2009.

[32] N. Tsingos, C. Dachsbacher, and S. Lefebvre, "Instant sound scattering," in *In Proc. of the 18th Eurographics Symposium on Rendering*. Grenoble, France, June 2007.

[33] N. Röber, M. Spindler, and M. Masuch, "Waveguide-based room acoustics through graphics hardware," in *Proc. International Computer Music Conference*. New Orleans, LA. USA, Nov. 2006.

[34] G. Campos and D. Howard, "A parallel 3d digital waveguide mesh model with tetrahedral topology for room acoustic simulation," in *Proc. COST G-6 Conference on Digital Audio Effects (DAFx-00)*. Verona, Italy, Dec. 2000.

[35] A. Southern, D. Murphy, G. Campos, and P. Dias, "Finite difference room acoustic modelling on a general purpose graphics processing unit," in *In 128th Audio Engineering Society Convention*. London, UK, May 2010.

[36] K. Kowalczyk and M. van Walstijn, "Modelling frequency-dependent boundaries as digital impedance filters in FDTD and K-DWM room acoustics simulations," *Journal of the Audio Engineering Society*, vol. 56, no. 7/8, pp. 569–583, 2008.

[37] Khronos Group, "OpenCL overview," http://www.khronos.org/opencl/, accessed April 6th, 2010.

[38] NVIDIA Corporation, "The next generation CUDA architecture," http://www.nvidia.com/object/fermi_architecture, accessed April 6th, 2010.

[39] J. O. Smith, *Digital Audio Resampling Home Page*, http://www-ccrma.stanford.edu/~jos/resample/, January 28, 2002.