Proceedings of the International Symposium on Room Acoustics, ISRA 2010

29–31 August 2010, Melbourne, Australia

Use of GPUs in room acoustic modeling and auralization

Lauri Savioja (1, 2), Dinesh Manocha (3), Ming C. Lin (3)

(1) Aalto University School of Science and Technology, Department of Media Technology, Espoo, Finland
(2) NVIDIA Research, Helsinki, Finland
(3) University of North Carolina, Chapel Hill, USA

PACS: 43.55.Ka

ABSTRACT

All room acoustic modeling techniques are computationally demanding due to the underlying complexity of sound propagation. Traditionally, all the modeling methods have been implemented for the central processing unit (CPU) of the computer. Recent trends in terms of programmable many-core processors or graphics processing units (GPU) is resulting in a paradigm shift. Modern GPUs are massively parallel processors with a different memory hierarchy, whose peak computational performance can be 10X more than that of current CPUs. In this paper, we give an overview of GPU architectures and address issues in designing suitable algorithms that map well to GPU architectures. The room acoustic modeling and auralization techniques are especially discussed in the light of possible GPU-based implementations. We also give a brief overview of recent methods for geometric and numeric sound propagation that offer one order of magnitude speedup over CPU-based algorithms.

INTRODUCTION

The traditional central processing units (CPU) are of a serial nature, and they are designed to perform operations in sequential order. Increase in CPU clock rates had guaranteed increasing performance of single-core computers for the past few decades, and performance of CPUs has closely followed the growth rate predicted by Moore's Law. The Moore's Law still holds and the amount of transistors on a single chip are still increasing. However, the clock rates are not increasing anymore due to the power and energy requirements. Instead, the main efforts in terms of improving the performance is to use multiple cores on the same chip, as opposed to making a single core faster.

The development of processors with multiple cores has led to two kinds of architectural trends. These include multi-core CPUs, which consist of best-performing serial cores. Examples of such processors include quad-core CPUs from Intel, AMD and other vendors. The second trend has been on design of heterogeneous architectures that combine fine-grain and coarse-grain parallelism using tens or hundreds of processors or cores. These processors are currently available as accelerators or many-core processors, which are designed with the goal of achieving higher parallel code performance. The demand for these accelerators is arising from consumer applications including computer gaming and multimedia. Examples of such accelerators include the graphics processors (GPUs), Cell processor and other data-parallel or streaming processors. Moreover, some recent industry announcements are pointing towards the design of heterogeneous processors and computing environments, which are scalable from a system with a single homogeneous processor to a high-end computing platform with tens, or even hundreds, of thousands of heterogeneous processors.

One of the most widely used example of a commodity many-core processor is the current programmable graphics processing unit (GPU). For example, current top-of-the-line GPUs from AMD/ATI or NVIDIA have tens or hundreds of simpler cores and high memory bandwidth, i.e. 10X more than current CPUs. This can result in much higher parallel code performance for a variety of applications. One of the very first applications that highlighted the potential of GPUs was the use of rasterization hardware to compute Voronoi diagrams in 2D and 3D and interactive motion planning in complex static and dynamic environments [1]. Over the last 11 years, computational power of GPUs has been exploited for scientific, database, geometric and imaging applications (i.e. GPGPU or use of GPUs for general purpose applications). In many cases, one order of magnitude performance gain was shown as compared to top of the line CPUs [2, 3]. The recent trend has been to use GPUs as co-processors for high performance computing and design high performance systems that can achieve sustained PetaFlop performance such as the recent Nebulae supercomputer [1].

COMPUTATION ON GPU

The current GPUs are regarded as high-throughput processors. They were initially designed and optimized to generate realistic images based on graphics rasterization algorithms. Over the last decade, the main trend has been in terms of GPUs becoming more general-purpose with a strong focus on achieving performance through high parallelism. Current high-end GPUs have a theoretical peak speed of up to a few Tera-FLOPs, thus far outpacing current CPU architectures. Specifically, there are several features that distinguish GPU architectures from the multi-core CPU systems and also make it harder to achieve peak performance. Still, the main use of GPUs is to generate realistic images. As a result, many of the architectural design features are mainly optimized for high rasterization performance, though the recent GPUs such as NVIDIA's Fermi architecture [4], include better support for general purpose parallel programming and double precision computation. In the rest of the section, we will review how the current GPUs operate.

[1] http://www.top500.org/lists/2010/06/press-release

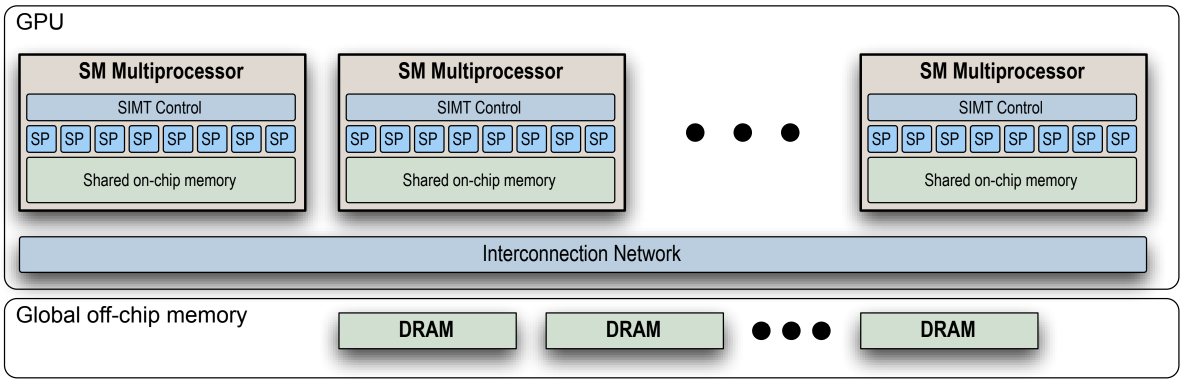

Figure 1: GPU Architecture: There are several streaming multiprocessors (SM) connected to each other with a fast interconnection network. Each SM has multiple streaming processors (SP) and a fast-access shared memory. A larger global memory with slower access is located off-chip.

GPU Architectures

Most of the modern GPUs operate on single-instruction multiple-data (SIMD) basis. Moreover, these computations are performed by simultaneously executing a high number of threads. A schematic illustration of a GPU structure is shown in Fig. 1. It shows that the GPU contains several streaming multiprocessors (SM) each of them containing a number of streaming processors (SP), a small shared memory and a control unit. All the SMs are connected to each other by a fast network. In addition, there is a global memory area that is accessible to all the SMs.

From performance point of view, a typical GPU memory system or hierarchy provides more bandwidth as compared to CPU memory systems, but has a higher latency. Moreover, only the very recent GPUs have a real two-level cache-hierarchy, though the GPU cache sizes are much smaller than CPU caches. In the older GPUs there was only a one-level read-only cache that was efficient in reducing the memory bandwidth but could not help that much in reducing the memory latency.

Program execution on GPU

Current GPUs have a relatively large number of independent processing cores (SMs in Fig. 1). Each of these cores is optimized to perform vector operations but runs at comparatively lower clock rate, as compared to current CPU cores. The high vector width - between 8 and 64 for current generation of GPUs - also implies that any efficient algorithm needs to utilize data parallelism to achieve high performance.

The programming of a GPU is done from the viewpoint of using multiple threads. Specifically, on current GPUs they are grouped into warps of multiple threads that are executed on a single SM (streaming multiprocessor). Inside one warp the processing is performed in a parallel manner, such that each thread issues the same instructions at a time, if any. This means that there is an immediate performance drop if there is any divergent branching inside a warp since it means that essentially all the threads perform all the branches. However, different SMs can have completely different execution patterns with no extra cost.

Each SM on the GPU can handle several separate tasks in parallel and switches between different tasks in hardware when one of them is waiting for a memory fetching operation to complete. This hardware multithreading approach is designed to hide the latency of memory accesses and used to perform some other work in the meantime. Thus, it is essential to have lots of threads in execution at once. In practice, there should be at least thousands of threads in one launch.

Programming tools

In the early days of GPGPU computation, the programming was performed using graphics APIs (application programming interface) such as OpenGL and DirectX. In practice, there were no constructs to support general computation and everything had to be posed as a graphics rasterization problem. A significant improvement happened when realtime programmable shaders were introduced on the UNC PixelFlow system [5] and were available on commodity GPUs in 2001 [6]. Only the introduction of CUDA (compute unified device architecture) in 2006 by NVIDIA brought the GPU computation to the mainstream and lead to the wide acceptance [7]. At the moment it is the most popular tool for GPU programming although it is a vendor specific API. An open alternative called OpenCL has been introduced lately by the Khronos Group and is supposed to provide a vendor-independent interface that should make it possible to program all future generations of GPUs [8].

Performance considerations

At a general level, it is good to remember that the maximum speedup achievable with parallelization is dictated by the Amdahl's law [9]:

$$S_{max} = \frac{1}{(1-P)+(P/N)} \tag{1}$$

where P is the fraction of computation that is parallelizable, and N is the number of parallel processors. The formula shows that if there is a large non-parallelizable part $(1-P)$ it starts to dominate the maximum speedup. On current GPUs N is already quite large, in the order of hundreds, and taking full advantage of them requires that P is close to one, in practice well above 0.9. To have even theoretical possibility for a 100-fold gain P must be at least 99%. This means that not all computational tasks or algorithms suit well for GPU computation, and it is worth considering only such problems that can be parallelized to numerous independent tasks, often cited as *embarrassingly parallel problems*.

The architectural characteristics of current GPUs impose some challenges on the programmers or application developers. In particular, the two main governing factors in terms of high GPU performance are [10]:

1. Provide a sufficient number of parallel tasks so that all the cores are utilized;
2. Provide several times that number of threads for each core. This is useful when some of the threads are waiting for data from relatively slow memory accesses, and the core is able to execute the instructions in another thread.

In addition, the GPU memory hierarchy affects the achievable performance. The on-chip shared memory seen of Fig. 1 can be used as cache, but it can be utilized directly as well. This on-chip memory is much faster than the global memory, and it should be utilized if maximum performance is targeted. However, use of that memory, beside cache usage, needs special attention and more skills of the programmer. To learn more about general performance optimization on GPUs see e.g. [7].

RAY-BASED MODELING AND GPU

In ray-based room acoustics, also called geometric acoustics (GA), sound is supposed to act as rays and the wavelength of sound is considered to be zero. This means that sound would propagate in straight lines and some other wave-phenomena, such as interference, are not modeled at all. This approach can be valid for high frequencies, and is widely used in room acoustic prediction. In these modeling techniques the main issues to be modeled are:

- Sound propagation in air
- Specular reflections
- Diffuse reflections
- Edge-diffraction

In the list above, the edge-diffraction is an exception to the GA approach as diffraction is actually a wave-based phenomenon. However, it has been noted that to make reliable simulations it has to be incorporated by some means. There are several different techniques to model edge-diffraction but the most commonly applied ones in room acoustics are the Biot-Tolstoy-Medwin (see, e.g. [11]) and the Kirchhoff approximations (see, e.g. [12]). To be even more exact, it is worth noting that diffuse reflection is neither a ray-based phenomenon, but caused by non-local reaction on the surface and by edge-diffraction of surface irregularities.

Ray-based modeling techniques

Ray-based acoustic modeling dates back to late 60's when the acoustic ray-tracing was introduced by Krokstad et al. [13]. There are several alternative modeling techniques based on this approach. For a long time these methods concentrated on finding the specular reflections in the room. One widely used GA technique is the image-source method [14]. It is accurate and is guaranteed to find all the reflection paths whereas ray-tracing statistically samples the path space and as the number of rays increases it asymptotically converges to the accurate solution.

Beam-tracing is an acceleration technique of the basic image-source method [15], [16]. It is still an accurate method, but avoids most of the geometrical intersection computations of the original image-source method. Laine et al. [17] have presented optimization techniques making it possible to compute image sources at interactive rates even in modest size geometries, but only for static scenes with fixed source locations. Another track of improvements to beam-tracing is formed by approximate methods that are not guaranteed to find all of the reflection paths. By this means it is possible to achieve remarkable performance gains as in the AD-frustum-tracing system that is able to handle even edge-diffraction with a uniform theory of diffraction (UTD) model [18]. These methods were demonstrated on multi-core CPUs, but it should be possible to port to many-core GPUs.

Ray-tracing has several further developed variants such as sonel mapping [19] and phonon mapping [20], and both these mappings are closely related to the photon mapping used in global illumination computation in computer graphics.

The room acoustic rendering equation is a unifying mathematical framework that covers all the GA modeling techniques [21]. Based on that equation, Siltanen et al. present a novel modeling technique called acoustic radiance transfer (ART). It is a generalized version of the radiosity method. In radiosity all the reflections are assumed to be diffuse such that in a reflection the outgoing direction of a ray is independent of the incoming direction whereas in ART there are so called BRDFs (bi-directional reflectance distribution functions) that determine the reflection characteristics of a surface more accurately.

Ray-tracing in computer graphics

Visual ray-tracing was one of the first applications to take advantage of the parallelism on the GPUs [22]. It is worth noting that GPUs have been designed and optimized to rasterize triangles, and traditionally ray-tracing was performed on CPUs. Ray-tracing is a good example of embarrassingly parallel problems, at least for static scenes where a hierarchy is precomputed. To make realistic images, tracing of thousands or even millions of rays is necessary and all of them are independent of each other and thus can be processed in parallel.

Over the last few years, there has been renewed interest in developing fast ray tracing algorithms on GPUs [23]. The same principles are applicable also in acoustic ray-tracing. In practice, all the ray-tracing systems use some spatial data structure to subdivide the space into smaller volumes or use bounding volume hierarchies (BVHs). For dynamic scenes, these hierarchical data structures can also be computed in parallel using GPU architectures [24, 25]. This implies that it is possible to reduce the number of required intersection computations between rays and polygons forming the space. Moreover, the actual ray tracing is performed at two levels. On the first one, the rays are traced on the intermediate nodes of the hierarchy. On the second level, the intersections between the ray and the polygons in one subvolume are computed. If the ray hits a surface, it gets reflected, and in the other case tracing is continued to the next subvolume. There are several alternatives in selection of the most suitable data structure. For dynamic scenes, the BVHs can be easily updated and therefore, can offer improved performance. In addition, it is essential to organize the ray traversal on GPU such that inside one warp the threads have maximally similar execution patterns [26]. However, it is not necessary to program a ray-tracer from scratch any more as there are existing highly optimized ray-tracing libraries for GPUs such as the OptiX by NVIDIA [27].

Use of GPUs in geometrical acoustics

There are several operations in GA models that can be parallelized. Below we will concentrate on the following three operations that are often needed in modeling:

- Geometrical computations
- Wave-based diffraction computations
- Convolutions

Accelerating ray-tracing

The first studies in which GPUs were used in acoustic ray-tracing date back to 2004 and 2007 [28, 29]. After that there has been lots of progress in visual ray-tracing as discussed above, and there are remarkable possibilities to speed-up acoustic ray-tracing as well.

Geometric models in graphics tend to be several orders of magnitude more complicated than typical acoustic models. One reason for this has been the limited computation capacity, and now introduction of GPU ray-tracing enables use of much more complex models in acoustic modeling as well. However, use of more detailed models and thus smaller surfaces might degrade the accuracy of simulations since the ray assumption of sound is valid only when the surfaces are large compared to the sound wavelength. Thus use of smaller surfaces makes the valid bandwidth narrower and only the highest frequencies are predicted reliably unless edge diffraction is modeled appropriately. It is still to be investigated, what would be the optimal geometric accuracy in ray-based room acoustic modeling.

The basic ray-tracing and image source methods are very similar to each other from computational point-of-view. The most time consuming part in both of them is reflection path traversal, and finding the intersections between those paths and surfaces on the model. One new study presenting use of GPU-based acoustics ray-tracing with highly complex models has been presented by Taylor et al. in the i-Sound sound rendering system [30]. This system uses a modified image-source algorithm. It shoots a fixed number of rays from each image-source to compute the visible geometric primitives. These visible primitives are used to compute higher order image sources based on specular reflections and edge diffraction. However, the computation of propagation paths from a source to the listener is decoupled from visible surface computation. In order to obtain higher performance, a novel multi-viewpoint ray tracing algorithm is used to compute the visible surface from each image-source in parallel on current GPUs. Given the fact that the overall approach decouples visibility computation from propagation paths, this approach needs to trace relatively fewer rays as compared to prior ray tracing based GA algorithms. The resulting parallel image source computation algorithm is able to trace more than 20 million rays/second on a NVIDIA GTX 280 GPU from different viewpoints to perform visibility computations. The performance of the propagation algorithm is directly proportional to the number of rays that can be traced in parallel. At a given resolution, the overall propagation algorithm is able to compute most of the propagation paths from the source to the listener, i.e. more than 95% of the paths for 3 orders of reflection and one order of diffraction. The accuracy is very high for the direct contributions and 1st order reflections, and decreases with the higher order of reflections. Moreover, i-Sound can handle dynamic scenes (with moving objects, sound sources or listener positions) that are commonly used in computer games at interactive rates (20-30fps).

Diffraction computation and GPU

Incorporation of diffraction into GA models is an essential factor in making them reliable at lower frequencies. Tsingos et al. have presented an efficient technique to compute first-order reflection and diffraction with the Kirchhoff approximation [31]. They are able to update the simulation results at interactive rates even in complex models containing almost 100 000 triangles. However, this model is only for first-order reflections, and to produce full-length impulse responses some other technique should be used to complement the response. Taylor et al. [30] also provide an approximate edge-diffraction algorithm based on ray tracing and UTD model. It has been used to perform first order diffraction on models with tens of thousands of triangles at interactive rates on current GPUs. It is worth noting that in the wave-based methods there is no need for separate treatment of diffraction since they model diffraction inherently.

Convolution and GPU

The frequency-domain acoustic radiance transfer is a technique suitable for real-time auralization [32]. In that method the number of required convolution operations is remarkable in complex models. Those are needed both in the energy transfer between surface patches and in the auralization. Computationally it is advantageous to perform those operations in the frequency-domain since in that domain a convolution is simply a point-wise multiplication of two vectors, and thus fully parallelizable for a GPU implementation.

WAVE-BASED MODELLING AND GPU

The most accurate room acoustic modeling results can be achieved with the wave-based modeling techniques that directly aim at solving the wave equation numerically. The most common methods in this category are the finite-difference time-domain (FDTD), finite-element (FEM) and boundary element (BEM) methods [33]. All of them are parallelizable and thus attractive for GPU implementation. Another advantage of wave-based modeling is that those methods typically solve the equation in the whole space whereas the GA methods most often compute the solution only to predetermined listener locations.

It has been shown that with the use of GPUs, even the boundary element method can be used to solve acoustic problems of realistic size [34]. Even more impressive results can be expected if the fast multipole variant (FMM) of the BEM has been implemented in the acoustic domain [35].

Finite-difference time-domain simulation

Digital waveguide mesh is a variant of the FDTD technique having its background in digital signal processing [36, 37]. In one of the first studies regarding use of GPUs in acoustics, the digital waveguide mesh was successfully accelerated by using a GPU to update the mesh [38]. They report close to 70-fold performance gains at best when compared to a pure CPU implementation.

In a more recent study, the FDTD simulation has been shown to be capable of real-time simulation of room acoustics at low and mid-frequencies [39]. In their system there is no separate convolution operation as the dry audio signal is directly fed into the FDTD mesh. The output signal is picked from the mesh node determined by an arbitrarily moving listener. In a model consisting of 150,000 nodes the achieved update rate was 7kHz. This amount of nodes is sufficient to model a living-room sized space discretized with 8cm regular grid. However, the FDTD methods suffer from dispersion that limits the valid bandwidth of simulations, thus in practice the result of that 7kHz simulation is usable only up to ca. 1kHz.

It is worth noting that in all wave-based methods using a volumetric grid, such as the FDTD and FEM, the computational load grows as a function of the fourth power of the grid spacing. This means that doubling the frequency band induces 16-fold amount of computation. If the currently reached 1kHz valid bandwidth would be expanded to 20kHz, the grid spacing should be only $1/20^{th}$ of the current spacing thus increasing the load by a factor of $20^4 = 160\,000$. However, it is possible to apply some off-line processing, such as frequency warping, to the resulting impulse response to decrease the dispersion and thus increase the valid bandwidth without increasing the FDTD mesh density [40]. In any case, covering the whole audible band with an FDTD simulation in real-time is still quite far away. At the moment it is possible to compute impulse responses off-line for concert-hall sized spaces valid up to 1kHz. For example, the volume of the Amsterdam Concertgebouw hall is ca. 21,000m^3, and if that is disrcetized with a 7cm grid corresponding to 8kHz update rate, the amount of nodes is around 50,000,000. Handling that on a GPU-based FDTD solver is not a problem, and a two second impulse response computation with frequency-independent boundary conditions would take only a few minutes. Adding more realistic frequency-dependency to the model increases the computation time. The actual increase depends on the proportion of the boundary nodes and of the complexity of the boundary, but in practical cases the increase is reported to be below 50% [39]. As the GPUs still improve rapidly, this computation is most likely to still get faster in the near future.

Adaptive Rectangular Decomposition (ARD)

Recently, Raghuvanshi et al. [41, 42] have introduced the adaptive rectangular decomposition approach to numerically solve the acoustic wave equation. This approach assumes a homogeneous medium in which the speed of sound is constant. The main benefit of ARD is in terms of dispersion errors. Numerical errors in wave equation solvers arise from discrete approximation of the differential operators in time and space. In case of FDTD algorithms, these errors manifest as numerical dispersion – all frequencies don't travel with the same speed and this leads to accumulative errors that eventually destroy the waveform being propagated. The ARD technique avoids such dispersion errors – by decomposing the scene into non-overlapping *rectangular* partitions, the spectral basis can be chosen to satisfy the Wave Equation directly. Assuming perfectly reflective walls for the rectangles, the basis functions turn out to be Cosine functions and it is possible to compute a closed form expression. The global solution is composed by coupling the interior solutions via sixth-order accurate, finite-difference transmission operators at the artificial interfaces. At the moment frequency-independent boundaries are supported in the ARD technique, and handling of frequency dependent boundaries still need further research.

The ARD computation algorithm has two primary stages: a) preprocessing the model, and b) numerical simulation to compute the impulse responses. In the preprocessing stage, the input scene is voxelized at the grid resolution and its size is determined by the maximum simulation frequency. This is followed by a rectangular decomposition step in which adjacent grid cells generated during voxelization are grouped into rectangles. These rectangles correspond to *air partitions*. Next, the algorithm generates artificial interfaces between adjacent rectangles and PML (perfectly matched layer) absorbing layers on the scene boundary, and uses them as *PML partitions*. This preprocessing is quite fast (1-2 minutes) for most scenes and takes less time than the overall simulation. The second stage of the algorithm corresponds to the numerical simulation that is performed on the grid. This involves computing a closed form solution within each partition and then updating the pressure field at the boundaries of the grid.

ARD on GPUs

Recently, Mehra et al. [43] presented a GPU-based algorithm to perform all the steps of numerical simulation of ARD on a GPU. The underlying algorithm exploits two levels of parallelism a) *coarse grained* and b) *fine grained*. Coarse grained parallelism is due to the fact that each of the partitions (air or PML) solves the wave equation in an independent manner. Fine grained parallelism is achieved because within each partition all the grid cells are independent of each other with regards to solving the wave equation at a particular time-step. Therefore within each partition all the grid cells can run in parallel exhibiting fine grained parallelism. The resulting GPU-based acoustic solver exploits both these levels of parallelism. It launches as many threads in parallel as there are partitions. Each of these threads performs the computations to solve the wave equation for a particular partition. All these threads are grouped into blocks and grids, and scheduled by the runtime environment on the GPU.

The overall GPU-based wave equation solver is implemented using the CUDA API. It is able to compute accurate impulse responses for complex scenes using single precision arithmetic.

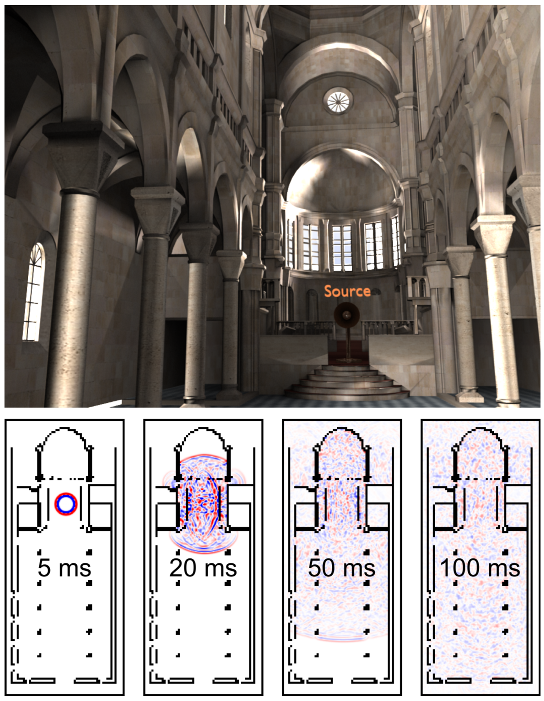

Figure 2: Sound simulation on a Cathedral. The dimensions of this scene are $35m \times 15m \times 26m$. The GPU-based ARD algorithm is perform numerical sound simulation on this complex scene on a desktop computer and pre-compute a 1 second long *impulse response* in about 250 seconds with the maximum frequency of 1,800Hz on NVIDIA Quadro FX 5800 GPU with 4GB memory.

The DCT and IDCT kernels are based upon the FFT library developed by Govindraju et al [44]. The synchronization between different threads is performed during interface handling and after each step of the simulation stage. The resulting solver was implemented on Nvidia Quadro FX 5800 graphics card with a core clock speed of 650 MHz, graphics memory of 4 GB with 240 CUDA cores. Its performance was also compared with a CPU-based implementation of the FDTD algorithm running on a quad-core Intel Xeon X5560 (4 cores) with processor speed of 2.8 GHz. It has been applied to many complex room models corresponding to a Cathedral (shown in Fig. 2), train station, and architecture models, whose volume varies between $7,000 - 83,000\text{m}^3$. The complexity depends on the number of cells generated by rectangular decomposition and they vary between $17 - 22$ millions cells, when the maximum simulation frequency is below 1500Hz. Each time-step of wave equation solver takes about 250ms on the GPU, and is about 10X faster than CPU-based ARD algorithm. Moreover, the GPU-based ARD algorithm can be up to 500X faster over prior CPU-based FDTD algorithms.

AURALIZATION AND GPU

A good overview on use of GPUs on auralization has been presented by Tsingos [45, 46]. In the context of this paper, the term auralization means the process of making the room acoustic modeling results audible. There are various techniques to implement auralization, but in practice the main workload is in audio signal processing. Especially, the most commonly needed operation is filtering. Basically there are two filter types: finite impulse response (FIR) or infinite impulse response (IIR) filters. The first of them is easily parallelizable for GPU [47, 48] whereas efficient parallel implementation of an IIR filter is still an open problem, although there have been proposals to improve their performance on GPU [49].

Convolution

One of the first applications of GPUs in auralization was sound spatialization for headphones. This head-related transfer-function (HRTF) computation has been studied for example by Gallo and Tsingos already in 2004 [50], and by Cowan and Kapralos [51, 52]. They were able to reach over 20-fold performance gain over CPU implementation. They all performed the convolution in the time-domain whereas Siltanen et al. use GPUs for HRTF convolution in the frequency domain [32].

Fourier transforms

As already mentioned, it is sometimes advantageous to perform the convolutions in the frequency-domain. For this reason, it is important to have an efficient fast Fourier transform (FFT) available. Utilization of GPUs in FFTs has been investigated a lot, and the results are impressive [44]. Savioja et al. have compared the performance of a GPU and CPU FFT libraries in real-time audio signal processing showing that a GPU is able to handle in real-time 8-times as long FFTs as a CPU-based implementation [48].

CONCLUSIONS

All the modern GPUs are suitable for parallel tasks such as ray-tracing or ray-frustum tracing. Current GPUs consist of hundreds of cores that can provide order of magnitude more computation power that current CPUs. In room acoustics they have been successfully utilized both in ray-based and wave-based methods. The achieved performance gains have been remarkable. The current trend in computing industry is to design multi-core and many-core processors and the number of cores are expected to increase at rate given by Moore's Law. As a result, all future algorithms for sound propagation and auralization need to exploit these parallel capabilities, see, e.g., [53, 54]. Audio signal processing is another area where GPUs can provide noticeable speedups. So far, HRTF filtering has been the main application of GPUs in the area, but any FIR filter would benefit from parallelization. However, it is worth noting that performing the signal processing operations on GPUs typically introduces some extra latency.

There are some challenges in use of GPU. Their memory hierarchy and performance is quite different than that of CPUs and one needs to design different kind of methods that can map well to GPUs. Furthermore, the on-chip memory on GPUs is limited (e.g. 4GB on current GPUs) and this can impose constraints on the size or volume of models that can be simulated. Finally, the programming environments for GPUs are quite different and the performance of the algorithms can vary considerably on different GPUs.

In additon to current GPUs, some future processors with a high number of cores may provide more flexibility and programmability. These include Intel's Many Integrated Core (MIC) architecture or AMD's Fusion processor, which could provide high throughput for acoustic modeling and auralization.

ACKNOWLEDGEMENTS

The authors would like to thank Dr. Timo Aila from NVIDIA Research for commenting the manuscript. This work was supported by the Academy of Finland (Project #130062). Dinesh Manocha and Ming C. Lin's work has been partially supported by Army Research Office, National Science Foundation and RDECOM.

REFERENCES

1 K. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha. Fast computation of generalized Voronoi diagrams using graphics hardware. *Proceedings of ACM SIGGRAPH*, pages 277–286, 1999.

2 J. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. Lefohn, and T. Purcell. A survey of general-purpose computation on graphics hardware. *Computer graphics forum*, 26(1):80–113, 2007.

3 D. Manocha. General-purpose computations using graphics processors. *Computer*, 38(8):85–88, 2005. ISSN 0018-9162. doi: http://dx.doi.org/10.1109/MC.2005.261.

4 NVIDIA Corporation. NVIDIA's next generation CUDA compute architecture: Fermi. http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf, accessed June 10th, 2010.

5 M. Olano and A. Lastra. A shading language on graphics hardware: The pixelflow shading system. *Proc. Of ACM SIGGRAPH*, pages 159–168, 1998.

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

7 NVIDIA Corporation. NVIDIA CUDA programming guide. pages 1–147, 2009.

8 Khronos Group. OpenCL overview. http://www.khronos.org/opencl/, accessed June 6, 2010.

9 G. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. *In Proc. of the AFIPS Spring Joint Computer Conf.*, pages 483–485, 1967.

10 C. Lauterbach, Q. Mo, and D. Manocha. gProximity: Hierarchical GPU-based operations for collision and distance queries. *Computer Graphics Forum (Proc. Of Eurographics)*, 28(2), 2010.

11 T. Lokki, U.P. Svensson, and L. Savioja. An efficient auralization of edge diffraction. In *Proc. AES 21st Int. Conf. on Architectural Acoustics and Sound Reinforcement*, pages 166–172, St. Petersburg, Russia, June 2002.

12 N. Tsingos, C. Dachsbacher, and S. Lefebvre. Instant sound scattering. *Proc. of the 18th Eurographics Symposium on Rendering*, 2007.

13 A. Krokstad, S. Strom, and S. Sorsdal. Calculating the acoustical room response by the use of a ray tracing technique. *J. Sound Vib.*, 8(1):118–125, 1968.

14 J. Allen and D. Berkley. Image method for efficiently simulating small-room acoustics. *Journal of the Acoustical Society of America*, 65(4):943–950, 1979.

15 U. Stephenson. Quantized pyramidal beam tracing - a new algorithm for room acoustics and noise immission prognos is. *ACUSTICA - acta acustica*, 82:517–525, 1996.

16 T. Funkhouser, I. Carlbom, G. Elko, G. Pingali, M. Sondhi, and J. West. A beam tracing approach to acoustic modeling for interactive virtual environments. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 21–32, New York, NY, USA, 1998. doi: http://doi.acm.org/10.1145/280814.280818.

17 S. Laine, S. Siltanen, T. Lokki, and L. Savioja. Accelerated beam tracing algorithm. *Applied Acoustics*, 70(1):172–181, 2009. Available at http://dx.doi.org/10.1016/j.apacoust.2007.11.011.

18 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*, 14(6):1707 – 1722, 2008.

doi: 10.1109/TVCG.2008.111.

19 B. Kapralos, M. Jenkin, and E. Milios. Acoustical modeling with sonel mapping. *Proc. 19th Intl. Congress on Acoustics (ICA)*, 2007.

20 M. Bertram, E. Deines, J. Mohring, and J. Jegorovs. Phonon tracing for auralization and visualization of sound. *IEEE Visualization*, 2005.

21 S. Siltanen, T. Lokki, S. Kiminki, and L. Savioja. The room acoustic rendering equation. *Journal of the Acoustical Society of America*, 122(3):1624–1635, 2007. doi: 10.1121/1.2766781.

22 N. Carr, J. Hall, and J. Hart. The ray engine. *In Proc. Graphics Hardware*, pages 1–10, September 2002.

23 S. Popov, J. Günther, H. Seidel, and P. Slusallek. Stackless KD-Tree Traversal for High Performance GPU Ray Tracing. *Computer Graphics Forum (Proc. EUROGRAPHICS)*, 26 (3):415–424, 2007.

24 K. Zhou, Q. Hou, R. Wang, and B. Guo. Real-time KD-tree construction on graphics hardware. *ACM Trans. Graph.*, 27 (5):1–11, 2008. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/1409060.1409079.

25 C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and Q. D. Manocha. Fast BVH construction on GPUs. *Computer Graphics Forum (Proc. Of Eurographics)*, 28(2):375–384, 2009.

26 T. Aila and S. Laine. Understanding the efficiency of ray traversal on GPUs. In *Proc. High Performance Graphics, HPG'09*. New Orleans, LA, USA, July 2009.

27 S. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich. Optix: A general purpose ray tracing engine. *ACM Transactions on Graphics*, August 2010.

28 M. Jedrzejewski and K. Marasek. Computation of room acoustics using programmable video hardware. In *Proc. International Conference on Computer Vision and Graphics, ICCVG 2004*, pages 587–592. Warsaw, Poland, September 2004. doi: 10.1007/1-4020-4179-9.

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

30 M. Taylor, A. Chandak, Q. Mo, C. Lauterbach, C. Schissler, and D. Manocha. i-Sound: Interactive gpu-based sound auralization in dynamic scenes. Technical report TR10-006, Computer Science, University of North Carolina at Chapel Hill, 2010.

31 N. Tsingos, C. Dachsbacher, and S. Lefebvre. Instant sound scattering. *In Proc. of the 18th Eurographics Symposium on Rendering*, June 2007.

32 S. Siltanen, T. Lokki, and L. Savioja. Frequency domain acoustic radiance transfer for real-time auralization. *Acta Acustica united with Acustica*, 95:106–117, 2009. doi: 10.3813/AAA.918132.

33 U. P. Svensson and U. Kristiansen. Computational modelling and simulation of acoustic spaces. In *Proc. AES 22nd Conf. on Virtual, Synthetic and Entertainment Audio*, pages 11–30. Espoo, Finland, June 2002.

34 T. Takahashi and T. Hamada. GPU-accelerated boundary element method for Helmholtz'equation in three dimensions. *International Journal for Numerical Methods in Engineering*, 80(10):1295–1321, 2009.

35 N. Gumerov and R. Duraiswami. Fast multipole methods on graphics processors. *Journal of Computational Physics*, 2008.

36 S. Van Duyne and J. Smith III. The 2-D digital waveguide mesh. *In Proc. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, New Paltz, NY, USA, October 1993.

37 L. Savioja, T. Rinne, and T. Takala. Simulation of room acoustics with a 3-D finite difference mesh. In *Proc. Int. Computer Music Conf.*, pages 463–466. Aarhus, Denmark, September 1994.

38 N. Röber, M. Spindler, and M. Masuch. Waveguide-based room acoustics through graphics hardware. *In Proc. International Computer Music Conference*, New Orleans, LA. USA, November 2006.

39 L. Savioja. Real-time 3D finite-difference time-domain simulation of low- and mid-frequency room acoustics. In *Proc. Int. Conference on Digital Audio Effects (DAFx-10)*, Graz, Austria, 2010.

40 L. Savioja and V. Välimäki. Interpolated rectangular 3-D digital waveguide mesh algorithms with frequency warping. *IEEE Transactions on Speech and Audio Processing*, 11(6): 783–790, Nov. 2003.

41 N. Raghuvanshi, B. Lloyd, N. Govindaraju, and M. Lin. Efficient numerical acoustic simulation on graphics processors using adaptive rectangular decomposition. *In Proc. EAA Symposium on Auralization*, Espoo, Finland, June, 2009.

42 N. Raghuvanshi, R. Narain, and M. Lin. Efficient and accurate sound propagation using adaptive rectangular decomposition. *IEEE Transactions on Visualization and Computer Graphics*, 15(5):789 – 801, 2009. doi: 10.1109/TVCG.2009.28.

43 R. Mehra, N. Raghuvanshi, M. Lin, and D. Manocha. Efficient GPU-based solver for acoustic wave equation. Technical report TR10-007, Computer Science, University of North Carolina at Chapel Hill, 2010.

44 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*, November 2008.

45 N. Tsingos. Using programmable graphics hardware for auralization. *In Proc. EAA Symposium on Auralization*, Espoo, Finland, June 2009.

46 N. Tsingos. Using programmable graphics hardware for acoustics and audio rendering. *In 127th Audio Engineering Society Convention*, New York, USA, October 2009.

47 A. Smirnov and T. Chiueh. Implementation of a FIR filter on a GPU. Technical report, ECSL, 2005.

48 L. Savioja, V. Välimäki, and J. Smith. Audio signal processing using graphics processing units. submitted to Journal of the Audio Engineering Society, 2010.

49 F. Trebien and M. Oliveira. Realistic real-time sound resynthesis and processing for interactive virtual worlds. *The Visual Computer*, (25):469–477, 2009.

50 E. Gallo and N. Tsingos. Efficient 3D audio processing with the GPU. *In Proc. ACM Workshop on General Purpose Computing on Graphics Processors*, August 2004.

51 B. Cowan and B. Kapralos. Spatial sound for video games and virtual environments utilizing real-time GPU-based convolution. *Future Play '08: Proc. 2008 Conference on Future Play: Research, Play, Share*, November 2008.

52 B. Cowan and B. Kapralos. Real-time GPU-based convolution: A follow-up. *In Proc. ACM FuturePlay@ GDC Canada, Intl. Conf. on the Future of Game Design and Technology*, 2009.

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

54 M. Taylor, A. Chandak, L. Antani, and D. Manocha. RESound: Interactive sound rendering for dynamic virtual environments. *In Proc. 17th International ACM Conference on Multimedia*, pages 1–10, July 2009.