Proc. of the EAA Symposium on Auralization, Espoo, Finland, 15-17 June 2009

USING PROGRAMMABLE GRAPHICS HARDWARE FOR AURALIZATION

Nicolas Tsingos

Sound Technology Research
Dolby Laboratories
San Francisco, USA
nicolas.tsingos@dolby.com

ABSTRACT

Over the last 10 years, the architecture of graphics accelerators (GPUs) has dramatically evolved, outpacing traditional general purpose processors (CPUs) with an average 2.25-fold increase in performance every year. With massive processing capabilities and high-level programmability, current GPUs can be leveraged for applications far beyond visual rendering.

In this paper, we offer an overview of modern programmable GPUs and how they can be applied to audio rendering. For applications ranging from sound synthesis and audio signal processing to numerical acoustic simulations, GPUs massive parallelism and dedicated instructions can offer a 5 to 100-fold performance improvement over traditional CPU implementations. We will illustrate such benefits with results from 3D audio processing and sound scattering simulations and discuss future opportunities for auralization on massively multicore processors.

1. INTRODUCTION

Driven by an increasing consumer demand for high quality interactive visuals, 3D graphics processors (GPUs) have dramatically evolved in the past decade. GPUs have moved from a specialized and pricy component only available on high-end professional workstations to a commodity component available on every consumer PC, reaching a market of hundreds of million units per year. At the same time, thanks to advances in manufacturing technology, GPUs evolved from ASICs implementing limited fixed-function processing to fully programmable, massively parallel processors capable of handling complex data structures. As can be seen in Figure 1, the raw computational power of current GPUs largely exceeds that of the most powerful general purpose processors (CPU) [1]. Arguably, GPUs are the first truly successful parallel processors.

As a result of this widespread availability and custom programming capabilities, modern GPUs have generated a lot of dedicated algorithmic research in the graphics community but also in many other areas in need of massive parallel performance. In today's games, GPUs are used not only to generate photorealistic 3D visuals but also accelerate visibility queries, character animation, path planning, artificial intelligence and physics calculations, including rigid and deformable body dynamics or fluid solvers. For a general overview of general purpose GPU (GP-GPU) applications, we refer the reader to the following references [2, 3, 4, 5, 6, 7].

Audio processing and rendering applications are among the most compute-intensive and often rely on additional DSP resources for real-time performance. However, programmable audio DSPs are often only available to product developers while consumer audio hardware (e.g., Creative Labs' SoundBlaster) generally implements fixed-function pipelines, which traditionally evolved at a rather slow pace. Increasingly, commodity audio hardware tends to disappear from console or PC architectures and audio processing tasks are left to the multiple available CPU cores. GPU features, such as multiply-accumulate instructions or multiple SIMD execution units, are similar to those of most DSPs [8, 9]. Moreover, 3D audio rendering and auralization [10, 11] not only require processing audio signals but also a significant number of geometric calculations, e.g. to determine sound occlusion or reflections, which are well suited to graphics architectures. As a result, GPUs are a compelling alternative to traditional DSPs for auralization.

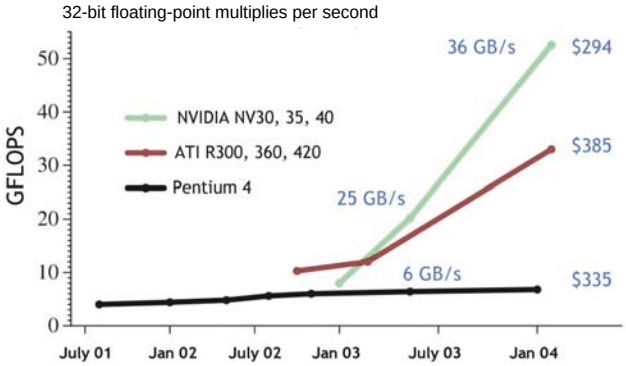

Figure 1: Evolution of GPU performance, memory bandwidth and commoditization. Courtesy of Pat Hanrahan and David Luebke.

In this paper, we review a number of recent contributions leveraging GPUs to accelerate audio rendering tasks ranging from audio signal processing and sound synthesis to numerical simulations of room acoustics and surface scattering. In Section 2, we give a short overview of the architecture and programming models of current GPUs, emphasizing some key differences between GPU and CPUs. To better highlight the architecture and programming model of GPUs and how they interact with the processing tasks, we chose to organize the remainder of the paper according to the different problems handled by auralization systems, which we can roughly break into three categories:

- Audio signal processing: in this first category the GPU processes an audio stream at typical audio sampling rates, in time or frequency domain, to perform a variety of processing, such as filtering, delay lines, synthesis, etc. We cover this application in Section 3.
- Numerical calculations and linear algebra: in this case, the GPU is used for numerical acoustics simulations, for in-

stance to solve large finite elements or finite differences problems. We review applications to room acoustics in Section 4.

- Geometry processing: in this final category, the GPU is used to process geometric primitives, such as polygons. This is the area where the highest gains can be expected compared to traditional CPU implementations since the calculations are closer to the original intent of graphics architectures. We review applications to acoustical ray-tracing, occlusion and scattering in Section 5.

Despite a number of successful implementations and applications, audio processing is not generally becoming a key component offloaded to the GPU. In section 6, we offer possible insights as to why this may be and what limits the use of GPUs for audio processing. Finally, the last couple of years have seen a convergence of CPU and GPU architectures with CPUs becoming more parallel and data paths between CPU and GPU becoming increasingly faster. In section 7, we discuss possible evolutions of integrated CPU and GPU programming and how this can benefit a variety of audio applications for auralization and beyond.

2. A QUICK TOUR INSIDE THE GPU

In this first section, we give a quick overview of the architecture of graphics accelerators and their recent evolution, focusing on the key differences with CPUs. A large body of work is available in the graphics community about GPU architecture and for additional detail we refer the reader to [12, 13].

2.1. The graphics pipeline and GPU architecture

Graphics hardware has a specific dataflow computational model. Its architecture is originally targeted at manipulating 3D primitives such as points, lines or polygons, performing some raster graphics operations and rendering the result to the screen. Primitives follow a sequence of operations before being processed to the screen. Figure 2 shows the essential steps of the pipeline.

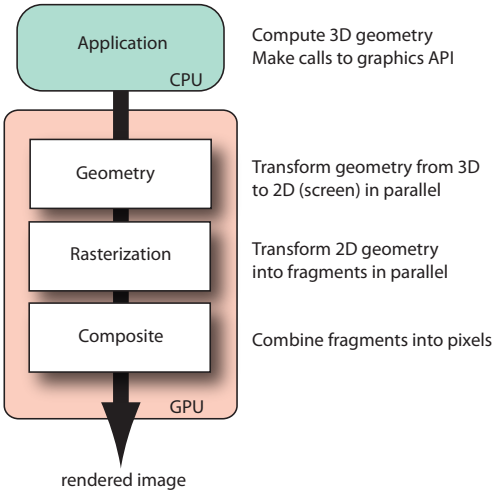

Figure 2: High-level overview of the graphics pipeline. In modern GPUs both the geometry and rasterization step contain user-programmable components called *shaders*.

The application transmits vertices of the primitives to the geometry stage. The vertex is a structure containing 3D position and texture coordinates, color and normal vector. The vertex processor applies any mathematical operation to each vertex including transformations from world space to screen space (i.e., camera projection). In the second step, vertices are assembled to form the geometric primitives (generally triangles). In this step, culling is computed to discard invisible surfaces according to their normal and the view direction. Next, clipping to the view frustum is applied before the rasterization. The view frustum is a set of planes which defines the field of view of the camera.

The second step of the pipeline rasterizes the transformed primitives. Rasterization determines which *fragment* of the screen buffer is covered by a primitive. The term fragment is employed instead of pixel since additional operations are still required to determine its final color. A fragment also carries additional characteristics such as color, normals or textures coordinates. These characteristics are interpolated for all fragments across the transformed vertices of the primitive. At this stage additional operations can be done to determine the final color of the fragment, e.g. local illumination models, cast shadows, etc. On recent hardware the fragment shading stage is fully programmable. Textures can also be applied to the fragments by sampling specific image buffers at non-integer coordinates. Several texture re-sampling schemes are provided by the hardware such as linear or bi-linear interpolation but any other scheme can be applied via the programming features of more recent GPUs.

Finally, the aim of the compositing stage is to perform additional tests before writing the final fragment values (i.e.,color) to the corresponding pixels of the frame buffer. Tests includes depth testing, used to remove hidden pixels and alpha test to handle transparency blending.

2.2. From fixed shading to general programming

Before the year 2000, most GPUs were fixed function and only available in high-end workstations (e.g., Silicon Graphics) for production or CAD-CAM applications. Starting in 2000, hardware accelerated graphics underwent a dramatic evolution in performance, features and commoditization. This evolution can be roughly split into three phases.

First, the fixed function pipeline became configurable allowing several textures to be combined and supporting off-screen rendering to a texture for multi-pass techniques. The ability to render into a texture and re-use the result for a subsequent rendering pass is one of the foundations of programmable graphics.

In 2002, programmable shading was introduced and allowed the programmer to write custom programs to transform vertices and compute the final appearance of a fragment (see Figure 3 top). Each program is executed by a large number of concurrent processors. The G70 family of Nvidia GPUs contained 24 parallel fragment pipelines. The power of the GPU comes from this data parallel architecture. *Circa* 2003-2004, another key element was the apparition of floating point support in the graphics pipeline in part motivated by the need for high-dynamic range rendering in computer graphics. This was key in evolving the GPU into a more general purpose processor.

Today, the separated programmable shading stages have been unified into a single model where a large number of small-footprint processing threads are executed by a global set of parallel processing cores (see Figure 3 bottom). The nVidia G80 GPUs, released

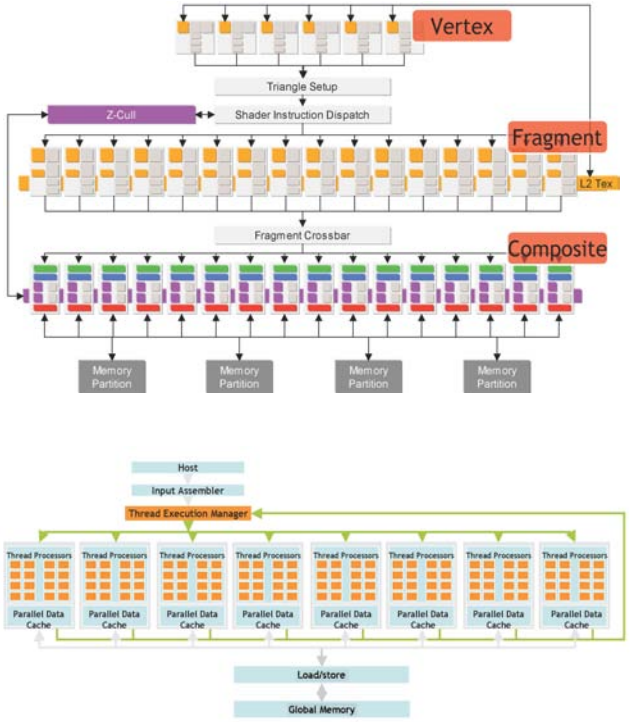

Figure 3: Evolution of GPU architecture. Top: architecture of a nVidia GeForce 6800 GPU, showing the separate vertex, fragment and compositing stages. Each stage is data-parallel and contains a large number of concurrent processing units. Bottom: Unified architecture on recent GeForce 8800 GPUs and beyond. Thread processors are dynamically allocated to vertex or fragment operations. Courtesy of Nick Triantos and Ian Buck, nVidia.

in 2007, contains 128 such stream processors which, in the case of graphics rendering, are automatically attributed to vertex or fragment processing. The latest GeForce GTX 295 features two onboard GPUs and a total of 480 processing cores. The latest hardware also supports an additional stage after the vertex shader, the *geometry shader* which can be used to create new geometry. It can also operate on primitives, as opposed to the vertex shader where vertex adjacency information is unknown.

Alongside with hardware evolution, programming tools have appeared to help developers leverage the capabilities of GPUs. Traditionally, graphics programming is achieved through two major APIs: *OpenGL* and *Direct 3D* [12]. It is now possible to program the GPU with high level C-like languages (e.g., Cg, GLSL, HLSL) [14, 15, 16] integrated within these APIs. The primary application of such tools is to create complex appearance models for surfaces, including shading and lighting operators.

However, for GP-GPU applications, other languages have recently been developed such as CUDA and recently openCL and the upcoming DirectX 11 compute shaders [17], that let the developer access the GPU as a stream processor without going through a graphics-oriented rendering pipeline. The stream programming model exposes the parallelism and communication patterns inherent in the application by structuring data into streams and expressing computation as arithmetic kernels that operate on streams [18, 19, 2, 20]. CUDA also includes fast Fourier transform (FFT) and linear algebra libraries optimized for *nVidia* GPUs (see also [21, 22]). CUDA has generated a lot of applications for GP-GPU programming [17], e.g. GPUMat, a library accelerating MATLAB code on the GPU and available as a freeware [23]. GPUFFTW also provides an FFT routine $4\times$ faster than the Intel Math Kernel Library on high-end Quad cores [24, 25, 26] (see Figure 6 bottom).

2.3. Differences between CPUs and modern GPUs

A major difference between CPUs and GPUs is that CPUs are optimized for high performance on sequential code so many of their transistors are dedicated to supporting non-computational tasks like branch prediction and caching. The highly parallel nature of graphics rendering enables GPUs to use additional transistors for computation achieving higher arithmetic intensity with the same transistor count. As a result, GPU threads are lighter but less efficient than CPU threads and require lots of parallelism and compute intensive tasks for performance. However, the computational throughput of GPUs also comes with limitations. While GPU shaders/kernels can *gather* data (e.g., they can access textures), they cannot easily write/*scatter* data to different memory locations[1] Hence, GPUs are better suited for programs that ideally have no data dependency allowing independent cores to process the data in parallel. Another implication is that GPUs are good at traversing data structures but very bad at building them.

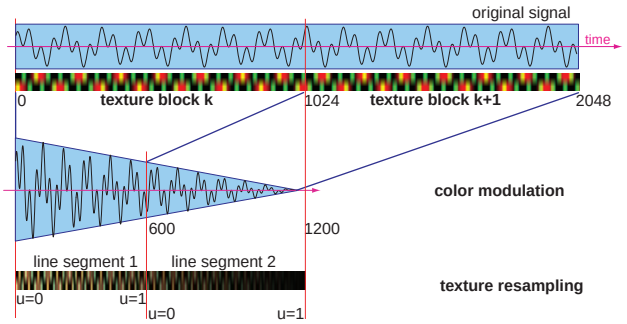

Figure 4: Processing audio with graphics APIs. In this example each pixel encodes a time-domain sample. Color channels are used to encode different frequency subbands. Audio samples are processed by drawing textured lines. Color modulation and texture resampling are used for equalization and pitch-shifting.

3. USING THE GPU FOR AUDIO SIGNAL PROCESSING

In this section, we review several applications of GPUs to audio processing and synthesis.

3.1. 3D audio processing and filtering

A first use of the GPU for auralization is to perform digital signal processing and filtering. Audio signal processing can be implemented through standard graphics APIs [27, 28] or in a more

[1] although this is somewhat possible at the level of the vertex and geometry shaders since their output can affect where the fragments will eventually be rendered.

straightforward manner using the recent general purpose APIs, described in Section 2.2. Several commercial sound processing plugins leveraging GPU processing are already available [29, 30, 31]. The latest release of Adobe Flash also provides integrated programming tools for GPU accelerated graphics and audio though the *Pixel Bender* API [32].

Graphics APIs being more standardized than GP-GPU APIs, using a graphics-oriented API ensures better compatibility among different hardware vendors. Processing audio through a graphics APIs involves converting the audio signal into textures and processing lines or polygons textured with the audio data. A variety of strategies has been used to store the audio data as textures. For instance, each pixel's color component can represent a single time-domain sample. RGBA components can also be used to store multiple subbands for efficient multi-band processing at the expense of more memory. Effects such as equalization or pitch-shifting can then be efficiently implemented through the use of accelerated vector instructions, e.g. dot-products, and texture resampling (Figures 4). Mixing can be achieved through floating-point blending in the compositing stage. Figure 5 illustrates an application to spatial audio rendering where the equalization coefficients are derived from Head Related Transfer Functions (HRTFs) to produce 3D sound over headphones [33, 10]. As can be seen in Figure 6 (top), this problem maps well on the GPU. With all audio samples pre-stored in graphics memory, the massive parallel architecture and fast memory bandwidth of modern GPUs delivers a 3-fold speedup over an SSE optimized implementation, in itself 20× faster than the original C code [28].

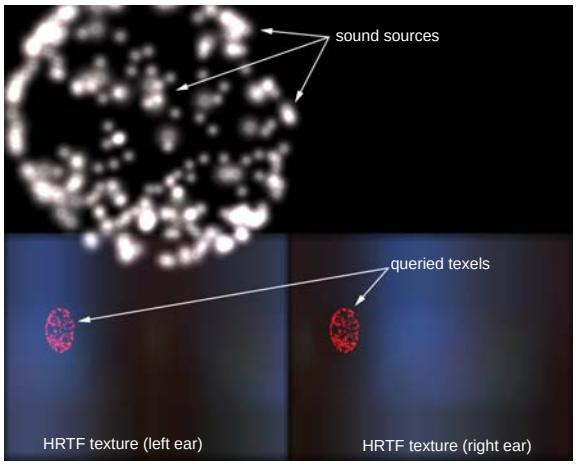

Figure 5: An illustration of a sphere of sounding particles auralized with the GPU. The input audio signals are equalized using an azimuth-elevation HRTF map. The intensity color of the RGBA components correspond to the attenuation for four different frequency subbands. The HRTF map is stored as a texture in graphics memory and can be efficiently queried using dedicated instructions. Queried samples in the maps are highlighted as red dots.

Alternatively, several contributions make use of GP-GPU APIs to perform fast finite impulse response (FIR) convolution, e.g. for reverberation effects. In that case, the GPU can process time-domain or frequency domain samples, performing the required multiply-accumulate instructions in parallel. As mentioned previously, the frequency domain transforms can also be directly implemented on the GPU [24, 21]. In [34], the authors report that the GPU outperforms an optimized SSE implementation for long FIR convolutions (more than 60000 taps) but fails for short filters. However, GPU implementations appear to bring significant speedups when compared to unoptimized CPU implementations [34, 35, 36]. With our own CUDA implementation of a Modified Cosine Transform (MDCT)-based reverberation engine, we observed a 6-fold speedup running on a Quadro FX3700 compared to a C++ version running on an Intel Core2 Extreme @3GHz.

Other algorithms such as infinite impulse response (recursive) filtering cannot be efficiently implemented on the GPU since past values are usually unavailable when rendering a given pixel in fragment programs. As suggested in [19], including persistent registers to accumulate results across fragments would solve this problem. Recently, Trebien and Oliveira have proposed a solution to implement general filtering on the GPU, including recursive feedback coefficients [37]. However, while the GPU has been shown to speed-up feedforward filters, they do not provide explicit timing comparisons between GPU and CPU for recursive filters.

An issue affecting performance for all signal processing applications on the GPU is the need for real-time streaming of data to the graphics memory. At audio sampling rates, streaming large numbers of audio streams will have a significant impact on the performance of the application due to the slow interconnects between GPU and main memory(8Gb/sec. on a PCI-E 16× bus vs. a typical 30Gb/sec. and up to 100Gb/sec. between GPU and graphics memory). As a result, speed-ups of 20× to 60× when processing data resident into graphics memory will tend to be reduced to 5× or 6× when audio has to be streamed.

3.2. Sound synthesis

A number of papers have also tackled the problem of synthesizing sound on the GPU, either by generating simple combinations of basic waveforms [38] or for physically-based synthesis, e.g. using modal models [37, 39, 40]. It should be noted that modal synthesis approaches are often implemented through recursive filtering and are likely to be less suited to a GPU implementation. Zhang et al., however, still reported 5× speed-ups compared to a CPU implementation in their experiments [39]. In the case of synthesis, no streaming to the graphics memory is required and approaches using little or no recursive filtering will tend to be more efficient, assuming that a large number of sounds are synthesized and mixed before being read-back to main memory.

4. USING THE GPU FOR NUMERICAL ACOUSTICS

GPUs can also be used to accelerate numerical acoustics, e.g. to solve finite elements, boundary elements or finite differences problems [41, 42, 43]. Röber et al. [41] implemented a room acoustic modeling tool on the GPU using digital waveguide meshes and report speed-ups of 4.5 to 69-fold in the 2D case, depending on the resolution of the lattice. For 3D simulations, they report more limited speed-ups which they attribute to the lack of 3D texture re-sampling in their test hardware. Current generation GPUs do offer 3D texture re-sampling and should provide speedups more in line with the 2D case. Recently, Raghuvanshi et al. [43] have introduced an efficient frequency-domain approach for large 3D environments by exploiting an adaptive rectangular decomposition of the domain. The approach heavily relies on the discrete cosine transform (DCT) which can be very efficiently computed on the GPU. This approach brings enormous speed-up from 90 to 300×

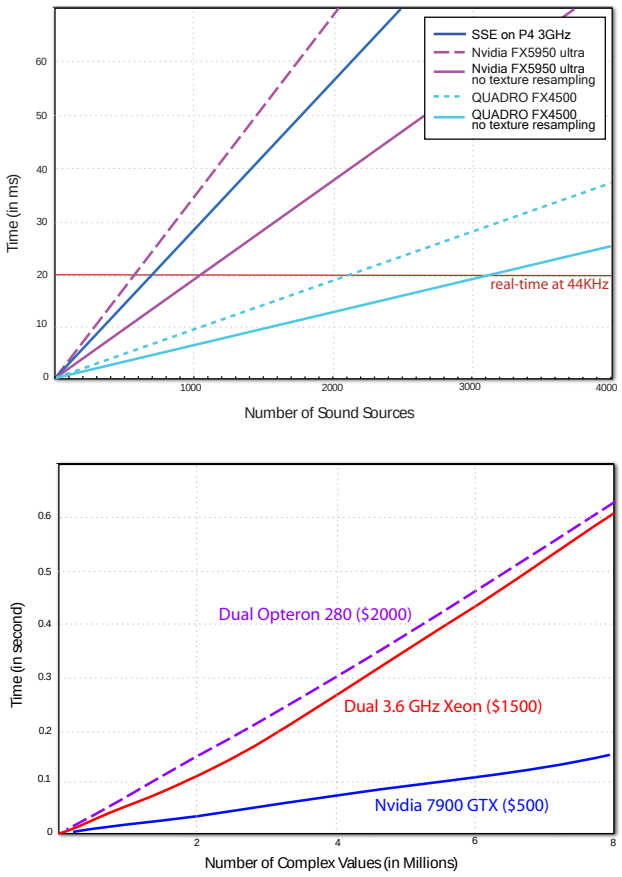

Figure 6: Performance comparison between GPU and CPU implementations. Top: A simple binaural renderer including 4-band equalization and a variable delay line [28]. Bottom: Massive 1D FFTs with GPUFFTW (from [24]).

when compared to an alternative finite difference time-domain approach running on the CPU.

Another popular numerical simulation technique, radiosity, computes radiant exchanges between surface patches and has been widely used in both computer graphics and acoustics. As far as we are aware, no solution has been proposed that leverages GPU processing to simulate acoustic radiant exchanges. However, in [44], the GPU is used in conjunction with the CPU to auralize the results of an acoustic radiance transfer simulation. In that case, the GPU is used to compute the accumulated response of all the surface patches in the environment and the final audio processing is performed on the CPU. Several approaches have been recently introduced in computer graphics to solve global illumination problems on the GPU that could certainly be adapted to sound propagation [45, 46, 47]. For instance, the *antiradiance* approach of [46], where visibility between surfaces is implicitly taken into account by propagating negative energy from backfacing surfaces shares many similarities with an acoustical boundary element approach.

5. GEOMETRY PROCESSING AND RASTERIZATION FOR AURALIZATION

Interactive auralization generally requires significant geometry processing, for instance to compute dynamic room acoustics or sound occlusion and scattering. Geometry processing typically includes sound source visibility queries and estimation of early sound reflections and diffraction.

5.1. Ray-casting for room acoustics

Ray-casting has been one of the earliest applications for programmable GPUs [48], demonstrating their ability to traverse data structures, such as trees, and bring significant speed-ups. GPU-accelerated ray-tracing has been used for auralization [49, 50] with significant 10-fold speedups reported. However, it is currently unclear how GPU approaches would compare against the latest advances in ray or beam-tracing [51], that leverage multiple CPU cores and vector instructions for ray intersections.

5.2. Occlusion and scattering

Several approaches have been proposed for determining sound occlusion and diffraction based on GPUs, even before graphics pipelines became programmable. These approaches share similarities with the *shadow mapping* techniques used in graphics to compute cast shadows. In 1997, Tsingos and Gascuel proposed to compute a qualitative sound occlusion factor by evaluating the amount of geometry blocking the first Fresnel ellipsoids defined by a source and listening points [52] using the GPU. At the time, the GPU could render the necessary occlusion map for a single source/listener pair at around 1 Hz. A similar recent approach [53] reports update rates reaching 2000 Hz.

Figure 7: Using graphics hardware for “sound visibility" calculations. (a) 3D view showing microphone, source, occluders and Fresnel zones for 400 and 4000 Hz. (b) Visibility map from microphone at 400Hz. (c) Visibility map from microphone at 4000Hz.

Another approach using the more physically grounded Kirchoff approximation (KA) was proposed in 1998 [54] and extended to support first-order reflections in 2007 [55]. This Kirchoff integral maps very well to direct rasterization by the graphics hardware, leading to a very efficient implementation. The approach renders all visible surfaces from the source(see Figure 8), similar to the reflective shadow-map algorithm of [56]. A complex fragment program computes the integrand for each pixel and hierarchical image averaging (*mip-mapping*) is used, in a second pass, to compute the final integral. Repeated rendering passes are used for different frequencies. A DirectX implementation can compute occlusions and first order reflections for two sources and 10 fre-

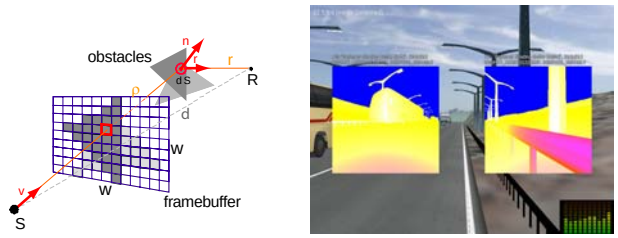

Figure 8: Left: Surfaces are sampled using hardware rendering from the point of view of the sound source. We evaluate the scattering terms at each pixel before global integration through mip-mapping. In this figure, S and R denote the source resp. receiver. Right: Visualization of the scattering terms on all surfaces visible from a sound source (here, the engine of a car).

quency subbands and also perform visual rendering at 100 Hz on a single GeForce FX8800.

Using the same formalism, the GPU can be used for computing higher-resolution scattering filters from very complex geometry, such as highly tessellated CAD-CAM models or those acquired through scanning techniques. Figure 9 and 10 illustrate scattering filters computed for large-scale real-world situations and compare the result to corresponding recordings taken in the field [2].

Figure 11 illustrates scattering impulse responses obtained from different surfaces. Such filters could be convolved along the propagation paths obtained with an image-source/beam-tracing technique or to model form-factors when obstacles are present between surface patches in radiosity algorithms.

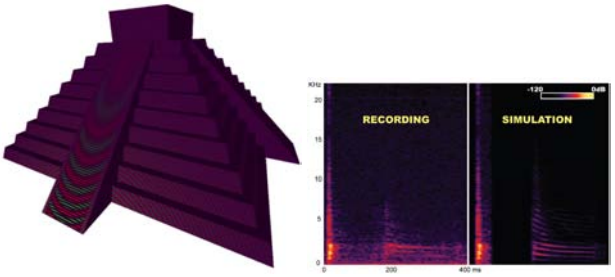

Figure 9: Left: Visualization of the scattering terms on the surface of a model of the Kukulkan temple, in Chichén Itzá, Mexico, for a 500Hz wave. The sound source is 15 meters in front of the stairs. Right: Comparison between spectrograms of a simulation and an on-site recording for the Kukulkan temple. The simulated response is convolved by the hand-clap of the original recording and convincingly captures the chirped echo from the stairs.

5.3. Acoustic reflectance and geometrical simplification

Interactive geometrical acoustics (GA) simulations can be enhanced by introducing diffraction effects from wedges. However, as all GA models, the geometrical theory of diffraction (GTD) assumes edges to be large compared to the wavelength. Increasing geometrical complexity would imply using smaller primitives and eventually would fall outside the validity domain of GA. Thus, it is

[2] Example audio files can be found at: http://www-sop.inria.fr/reves/projects/InstantScattering.

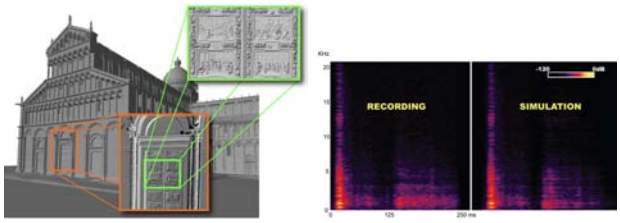

Figure 10: Left: A 3D model of the scanned façade of the Duomo in Pisa, Italy and close-ups on surface detail. Time-of-flight laser scanning was used to obtain this 13 million triangle model to a 2cm resolution. Right: Comparison between spectrograms of a simulation and an on-site recording. The simulated response is convolved by the hand-clap of the original recording.

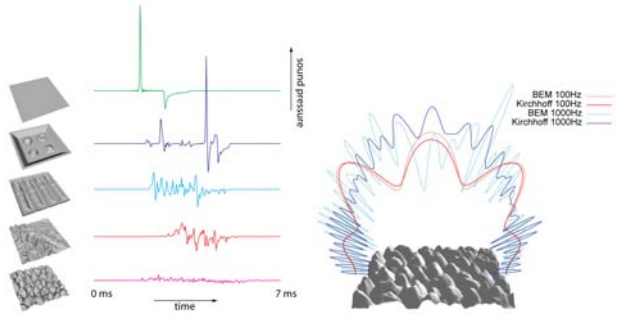

Figure 11: Left: Responses from different 4×4m surface samples. Each surface is composed of 131072 triangles and generated from displacement maps. Note the secondary scattering component due to the finite extent of the flat surface on the top row (green curve) and the increasingly “diffusing" nature of the surfaces from top to bottom. Right: Scattering patterns for a detailed surface. The figure compares sound pressure levels in a plane medial to the surface obtained by BEM and our approximation. Source is 5m directly above the center of the face and the pressure is plotted at a distance of 10m.

unclear how classical GA+GTD approaches could apply to more detailed scenes. Recent works have been devoted to level-of-detail (LOD) approaches for GA [57, 58, 59] but to our knowledge no general simplification scheme that preserves the correct scattering properties has been proposed to date.

Surface integrals and the use of GPUs offer the possibility to leverage level-of-detail schemes originally developed for appea–rance-preserving simplification in computer graphics [60, 61]. For instance, Tsingos et al. [62] proposed a strategy combining normal mapping with displacement correction to model complex surface detail for acoustic scattering calculations. Displacement surfaces [63, 64, 61] use textures to encode fine-grain surface detail which can be used at rendering time by a software ray-tracer or with the graphics hardware. Figure 12 shows scattering impulse responses calculated from a reference geometry and a flat polygon using normal and displacement textures. As can be seen, changes in the surface normal result in very little difference compared to a flat surface (compare to the top row in Figure 11). This demonstrates the importance of surface displacement and the resulting interference phenomena which are paramount in modeling the proper scattering effect. A displacement-corrected normal

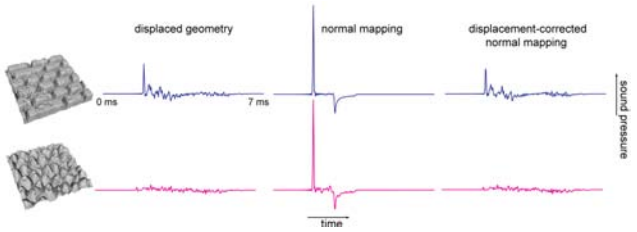

Figure 12: Comparison of true displaced geometry with a proxy flat quadrilateral enhanced with normal-map only or combined normal/displacement maps. Source and receiver are respectively 10 and 20 m directly above the center of the face. Note how the normal-map alone has little effect on the obtained response. The amplitude of displacement is 0.5 meters.

mapping results in a much better approximation.

An alternative solution would be to consider a more accurate edge-diffraction model, such as the Biot-Tolstoy-Medwin (BTM) model [65]. Contrary to the GTD, the BTM model can accurately model diffraction off finite-length edges. For finely tesselated meshes, a frequency-domain BTM approach [66] could certainly be accelerated on the GPU.

5.4. Individualized HRTF modeling

Most advanced auralization pipelines support individualized binaural rendering, using measured or parametric HRTFs in order to best match individual listeners [33, 67, 68]. Individualized HRTFs are traditionally obtained either through measurements [69] or numerical simulation [70, 71]. GPU accelerated ray-tracing or surface integrals can provide an efficient solution to individualized HRTF modeling, at a fraction of the computational cost of full boundary element simulations. Several approaches have been proposed toward this goal [49, 72] but no comparative study has been performed to date to evaluate the results.

6. DISCUSSION AND LIMITATIONS

For audio signal processing applications, GPU implementations generally do bring significant speedups over unoptimized C/C++ implementations. The gain is generally more limited when compared to SSE optimized implementations. although one could argue that producing optimized SSE assembly code is less straightforward than a GPU port using the available high-level APIs. For a number of key applications, games in particular, a very limited amount of spare GPU cycles is generally available and game developers have been reluctant to use the graphics processing resources for other applications, although most consoles features fast interconnections between GPU and CPU (20Gb/sec on Xbox360 and PS3). Other issues, such as inefficient recursive filtering might also affect the decision of porting audio processing pipelines to the GPU.

The key factor that currently limits GPU efficiency for audio processing is certainly the limited communication bandwidth between GPU and CPU, even with current PCI express buses. This impacts the efficiency of streaming audio data to graphics memory and reading-back of the processed streams for output. Approaches have been proposed to directly output audio data through the VGA port [73] but this is unlikely to become common practice, even through the HDMI output now available of most graphics cards. New interconnections such as PCI Express 3.0 or hypertransport with higher bandwidth might make GPUs better candidates for offloading audio processing tasks in the future.

For geometrical acoustics or occlusion/scattering applications, the GPU are very compelling alternatives, bringing much greater speedups and generally requiring slower and more limited asynchronous readbacks.

7. CONCLUSIONS AND FUTURE OPPORTUNITIES

We are certainly at a critical point in time where GPU architectures have opened the way to widespread parallel programming. With new processors, such as Intel's Larrabee [74, 75] and increased interconnect speeds, GPU and CPU will increasingly be working together supported by general purpose programming APIs [76]. It is clear that massively parallel architectures will offer tremendous benefits to audio and auralization applications.

As discussed in this paper, a variety of audio processing and computational acoustics algorithms have already been ported successfully to such architectures, sometimes twisting the original problem to recast it in terms of graphics programming. This issue increasingly disappears with new programming models and APIs and could lead to more widespread use for wavefield synthesis [77] or microphone array techniques [78], which require massive parallel processing. For acoustic design, massively parallel computing could enable fast goal-directed design and optimization of acoustic spaces or scattering surfaces, by allowing simulation algorithms to run in an interactive optimization loop [79]. GPUs could also allow for efficient audio coding/decoding and integrated transform-domain processing of a large number of audio streams for remote audio rendering and voice-chat applications. Beyond auralization, GPUs are also already used for speech recognition to compute acoustic likelihoods and hidden Markov models [80, 81, 82] and could contribute to making speech-driven interfaces more efficient, robust and widely accepted.

Finally, following the evolution of the rendering hardware, new algorithms and approaches have been introduced in the graphics community. Even more than before, we believe it is important to follow the progress of computer graphics approaches, some of which address problems very close to their acoustical counterparts. They could lead to significant advances in geometrical acoustic modeling, for instance for dynamic simplification and geometry processing or improved scattering models.

8. REFERENCES

[1] Magnus Ekman, Fredrik Warg, and Jim Nilsson, "An in-depth look at computer performance growth," *SIGARCH Comput. Archit. News*, vol. 33, no. 1, pp. 144–147, 2005.

[2] D. Geer, "Taking the graphics processor beyond graphics," *Computer*, vol. 38, no. 9, pp. 14–16, Sept. 2005.

[3] Avi Bleiweiss, "GPU accelerated pathfinding," in *GH '08: Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, Aire-la-Ville, Switzerland, Switzerland, 2008, pp. 65–74, Eurographics Association.

[4] "GPGPU - General-Purpose Computation on Graphics Hardware," Available at http://www.gpgpu.org, accessed May 15, 2009.

[5] "GPGPU - Tutorial and Courses.," Available at http://gpgpu.org/tag/tutorials-courses, Accessed May 15, 2009.

[6] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn, and Timothy J. Purcell, "A survey of general-purpose computation on graphics hardware," *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, 2007.

[7] M. Pharr, Ed., *GPU Gems 2: Programming Techniques, Tips and Tricks for Real-Time Graphics, Part IV: General-Purpose Computation on GPUS: A Primer*, Addison-Wesley Professional, 2005, Available at http://http.developer.nvidia.com/GPUGems2/gpugems2_part04.html.

[8] P. Lapsley, J. Bier, A. Shoham, and E.A. Lee, *DSP Processor Fundamentals*, IEEE Press, 1997.

[9] J. Eyre and J. Bier, "The evolution of DSP processors," *IEEE Signal Processing Magazine*, 2000, See also http://www.bdti.com/.

[10] Durand R. Begault, *3D Sound for Virtual Reality and Multimedia*, Academic Press Professional, 1994.

[11] L. Savioja, J. Huopaniemi, T. Lokki, and R. Väänänen, "Creating interactive virtual acoustic environments," *J. Audio Eng. Soc.*, vol. 47, no. 9, pp. 675–705, Sept. 1999.

[12] D. Shreiner, M. Woo, J. Neider, and T. Davis, *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 2.1 (6th Edition)* , Addison-Wesley, 2007.

[13] David Luebke and Greg Humphreys, "How GPUs Work," *Computer*, vol. 40, no. 2, pp. 96–100, Feb. 2007.

[14] R. Rost, *OpenGL(R) Shading Language (2nd Edition)* , Addison-Wesley, 2006.

[15] William R. Mark, R. Steven Glanville, Kurt Akeley, and Mark J. Kilgard, "Cg: a system for programming graphics hardware in a c-like language," in *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, New York, NY, USA, 2003, pp. 896–907, ACM.

[16] Randima Fernando and Mark J. Kilgard, *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*, Addison-Wesley Professional, 2003.

[17] "Nvidia CUDA," Available at http://www.nvidia.com/cuda, accessed May 15, 2009.

[18] I. Buck and T. Purcell, *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics. Chapter 37: A Toolkit for Computation on GPUs*, Addison-Wesley Professional, 2004, Available at http://http.developer.nvidia.com/GPUGems/gpugems_ch37.html.

[19] Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, and Pat Hanrahan, "Brook for GPUs: stream computing on graphics hardware," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 777–786, 2004.

[20] John Owens, *GPU Gems 2: Programming Techniques, Tips and Tricks for Real-Time Graphics. Chapter 29: Streaming Architectures and Technology Trends*, Addison-Wesley Professional, 2005, Available at http://developer.nvidia.com/object/gpu_gems_2_home.html.

[21] Kenneth Moreland and Edward Angel, "The FFT on a GPU," in *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, Aire-la-Ville, Switzerland, Switzerland, 2003, pp. 112–119, Eurographics Association.

[22] Jens Krüger and Rüdiger Westermann, "Linear algebra operators for gpu implementation of numerical algorithms," in *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, New York, NY, USA, 2003, pp. 908–916, ACM.

[23] "GPUmat: A GPU toolbox for MATLAB," Available at http://gp-you.org/, accessed May 15, 2009.

[24] "GPUFFTW: High Performance Power-of-two FFT library using graphics processors," Available at http://gamma.cs.unc.edu/GPUFFTW/, accessed May 15, 2009.

[25] Naga K. Govindaraju and Dinesh Manocha, "Cache-efficient numerical algorithms using graphics hardware," *Parallel Comput.*, vol. 33, no. 10-11, pp. 663–684, 2007.

[26] Naga K. Govindaraju, Scott Larsen, Jim Gray, and Dinesh Manocha, "A memory model for scientific algorithms on graphics processors," Nov. 2006, pp. 6–6.

[27] S. Whalen, "Audio and the Graphics Processing Unit," Available at http://www.node99.org/papers/gpuaudio.pdf, accessed May 15, 2009.

[28] E. Gallo and N. Tsingos, "Efficient 3D audio processing with the GPU," in *Proc. ACM Workshop on General Purpose Computing on Graphics Processors (poster), Los Angeles,*, Aug. 2004, http://www-sop.inria.fr/reves/projects/GPUAudio/.

[29] LiquidSonics, "Reverberate LE GPU Edition," Available at http://www.liquidsonics.com/software.htm, accessed May 15, 2009.

[30] Nils Schneider, "VST Plugin: Convolution Reverb on NVidia GPUs ," Available at http://www.nilsschneider.de, accessed May 15, 2009.

[31] Acustica Audio, "Nebula 3 VST Plugin," Available at http://www.acusticaudio.net/, accessed May 15, 2009.

[32] "Adobe PixelBender API," Available at http://labs.adobe.com/technologies/pixelbender, accessed May 15, 2009.

[33] J. Blauert, *Spatial Hearing : The Psychophysics of Human Sound Localization*, M.I.T. Press, Cambridge, MA, 1997.

[34] A. Smirnov and T. Chiueh, "An Implementation of a FIR Filter on a GPU," Available at http://research.alexeysmirnov.name/index.php?area=gr&proj=fog, accessed May 15, 2009.

[35] B. Cowan and B. Kapralos, "Real-time GPU-base convolution: A follow-up," in *Proc. of the FuturePlay @ GDC Canada Intl. Conf. on the Future of Game Design and Technology*, May 12-13 2009.

[36] Brent Cowan and Bill Kapralos, "Spatial sound for video games and virtual environments utilizing real-time GPU-based convolution," in *Future Play '08: Proceedings of the 2008 Conference on Future Play*, New York, NY, USA, 2008, pp. 166–172, ACM.

[37] F. Trebien and M.M. Oliveira, "Realistic real-time sound resynthesis and processing for interactive virtual worlds," *The Visual Computer*, vol. 25, no. 5-7, pp. 469–477, May 2009.

[38] F. Trebien and M.M. Oliveira, *Real-time Audio Processing on the GPU*, pp. 583–604, Charles River Media, 2008.

[39] Q. Zhang, L. Ye, and Z. Pan, "Physically-based sound synthesis on GPUs," in *Proc. of the 4th Intl. Conf. on Entertainment Computing*, Sept. 19-21 2007.

[40] Christoph von Tycowicz and Jörn Loviscach, "A malleable drum," in *SIGGRAPH '08: ACM SIGGRAPH 2008 posters*, New York, NY, USA, 2008, pp. 1–1, ACM.

[41] N. Röber, M. Spindler, and M. Masuch, "Waveguide-based Room Acoustics through Graphics Hardware," in *Proc. Intl. Computer Music Conf. (ICMC)*, Nov. 6-11, 2006.

[42] N. Raghuvanshi, B. Lloyd, and M.C. Lin, "Efficient Numerical Acoustic Simulation on Graphics Processors Using Adaptive Rectangular Decomposition," in *Proc. EAA Symp. on Auralization*, June 15-17, 2009.

[43] Nikunj Raghuvanshi, Rahul Narain, and Ming C. Lin, "Efficient and accurate sound propagation using adaptive rectangular decomposition," *IEEE Transactions on Visualization and Computer Graphics*, vol. 99, no. 2, 5555.

[44] Samuel Siltanen, Tapio Lokki, and Lauri Savioja, "Frequency domain acoustic radiance transfer for real-time auralization," *Acta Acustica united with Acustica*, vol. 95, pp. 106–117(12), January/February 2009.

[45] Nathan A. Carr, Jesse D. Hall, and John C. Hart, "GPU algorithms for radiosity and subsurface scattering," in *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, Aire-la-Ville, Switzerland, Switzerland, 2003, pp. 51–59, Eurographics Association.

[46] Carsten Dachsbacher, Marc Stamminger, George Drettakis, and Frédo Durand, "Implicit visibility and antiradiance for interactive global illumination," in *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, New York, NY, USA, 2007, p. 61, ACM.

[47] Jaakko Lehtinen, Matthias Zwicker, Emmanuel Turquin, Janne Kontkanen, Frédo Durand, François Sillion, and Timo Aila, "A meshless hierarchical representation for light transport," *ACM Trans. Graph.*, vol. 27, no. 3, 2008.

[48] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan, "Ray tracing on programmable graphics hardware," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 703–712, July 2002, ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).

[49] N. Röber, U. Kaminski, and M. Masuch, "Ray-acoustics using Computer Graphics Technology," in *Proc. 10th Intl. Conf. on Digital Audio Effects (DAFx)*, Sep. 10-15, 2007.

[50] M. Jedrzejewski and K. Marasek, "Computation of room acoustics using programmable video hardware," in *Proc. Computer Vision and Graphics International Conference, ICCVG 2004, Warsaw, Poland*, September 2004.

[51] A. Chandak, C. Lauterbach, M. Taylor, Z. Ren, and D. Manocha, "AD-Frustum: Adaptive Frustum Tracing for Interactive Sound Propagation," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 14, no. 6, pp. 1707–1722, Nov.-Dec. 2008.

[52] Nicolas Tsingos and Jean-Dominique Gascuel, "Soundtracks for computer animation: sound rendering in dynamic environments with occlusions," in *Proc. of Graphics Interface'97*, May 1997, pp. 9–16.

[53] B. Cowan and B. Kapralos, "Real-time acoustical diffraction modeling using the GPU," in *Proc. of the 10th Western Pacific Acoustics Conf.*, Sept. 21-23 2009.

[54] Nicolas Tsingos and Jean-Dominique Gascuel, "Fast rendering of sound occlusion and diffraction effects for virtual acoustic environments," in *Proc. 104th Audio Engineering Society Convention, preprint 4699*, Amsterdam, Netherlands, May 1998.

[55] Nicolas Tsingos, Carsten Dachsbacher, Sylvain Lefebvre, and Matteo Dellepiane, "Instant sound scattering," in *Rendering Techniques (Proc. of the Eurographics Symposium on Rendering)*, 2007.

[56] C. Dachsbacher and M. Stamminger, "Reflective shadow map," *Proceedings of I3D'05*, 2005.

[57] S. Siltanen, "Geometry reduction in room acoustics modeling," *Master Thesis, Helsinki University Of Technology, Department of Computer Science Telecommunications Software and Multimedia Laboratory*, September 2005.

[58] L.M. Wang, J. Rathsam, and S.R. Ryherd, "Interactions of model detail level and scattering coefficients in room acoustic computer simulation," *Intl. Symp. on Room Acoustics, a satelite symposium of ICA, Kyoto, Japan*, 2004.

[59] C. Joslin and N. Magnenat-Thalmann, "Significant facet retreival for real-time 3D sound rendering in complex virtual environments," *Proc. of VRTST 2003*, October 2003.

[60] Robert L. Cook, Loren Carpenter, and Edwin Catmull, "The reyes image rendering architecture," *SIGGRAPH Comput. Graph.*, vol. 21, no. 4, pp. 95–102, 1987.

[61] Jonathan Cohen, Marc Olano, and Dinesh Manocha, "Appearance-preserving simplification," in *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, 1998, pp. 115–122, ACM Press.

[62] Nicolas Tsingos, Carsten Dachsbacher, Sylvain Lefebvre, and Matteo Dellepiane, "Extending geometrical acoustics to highly detailed architectural environments," in *19th Intl. Congress on Acoustics*, sep 2007.

[63] J. Hirche, A. Ehlert, S. Guthe, and M. Doggett, "Hardware accelerated per-pixel displacement mapping," *Proc. of Graphics Interface'04. Canadian Human-Computer Communications Society*, pp. 153–158, 2004.

[64] Lionel Baboud and Xavier Décoret, "Rendering geometry with relief textures," in *Graphics Interface '06*, 2006.

[65] U. P. Svensson, R. I. Fred, and J. Vanderkooy, "An analytic secondary source model of edge diffraction impulse responses," *J. Acoust. Soc. Am.*, vol. 106, pp. 2331–2344, 1999.

[66] U. P. Svensson, P. Calamia, and S. Nakanishi, "Frequency-domain edge diffraction for finite and infinite edges," *Acta Acustica united with Acustica*, vol. 95, no. 3, 2009.

[67] E. Wenzel, M. Arruda, D. Kistler, and F. Wightman, "Localization using non-individualized head-related transfer functions," *J. Acoustical Soc. Am.*, vol. 94, no. 1, pp. 111–123, July 1993.

[68] D. Begault, E. Wenzel, and M. Anderson, "Direct comparison of the impact of head-tracking, reverberation, and individualized head-related transfer functions on the spatial perception of a virtual speech source," *J. Audio Eng. Soc.*, vol. 49, no. 10, pp. 904–916, 2001.

[69] J.C. Middlebrooks, E.A. Macpherson, and Z.A. Onsan, "Psychophysical customization of directional transfer functions for virtual sound localization," *Journal Acoustical Soc. Am.*, vol. 108, no. 6, pp. 3088–3091, 2000.

[70] Y. Kahana and P.A. Nelson, "Numerical modelling of the spatial acoustic response of the human pinna," *Journal of Sound and Vibration*, vol. 292, no. 1-2, pp. 148–178, Apr. 2006.

[71] B. Katz, "Boundary element method calculation of individual head-related transfer function. part I: Rigid model calculation," *Journal Acoustical Soc. Am.*, vol. 110, no. 5, pp. 2440–2448, 2001.

[72] Matteo Dellepiane, Nico Pietroni, Nicolas Tsingos, Manuel Asselot, and Roberto Scopigno, "Reconstructing head models from photographs for individualized 3D-audio processing," in *Computer Graphics Forum (Special Issue - Proc. Pacific Graphics) 27(7)*, 2008.

[73] Jörn Loviscach, "GPU-based audio via the VGA port," in *SIGGRAPH '08: ACM SIGGRAPH 2008 posters*, New York, NY, USA, 2008, pp. 1–1, ACM.

[74] Brian Santo and Sally Adee, "Multi-core made simpler," *IEEE Spectrum*, Jan. 2009, Available at http://www.spectrum.ieee.org/jan09/7129.

[75] Larry Seiler, Doug Carmean, Eric Sprangle, Tom Forsyth, Michael Abrash, Pradeep Dubey, Stephen Junkins, Adam Lake, Jeremy Sugerman, Robert Cavin, Roger Espasa, Ed Grochowski, Toni Juan, and Pat Hanrahan, "Larrabee: a many-core x86 architecture for visual computing," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 1–15, 2008.

[76] M. Pharr, A. Lefohn, C. Kolb, P. Lalonde, T. Foley, and G. Berry, "Programmable Graphics - The Future of Interactive Rendering," Available at http://www.cerlsoundgroup .org/RealTimeMorph/, accessed March 08, 2006.

[77] A.J. Berkhout, D. de Vries, and P. Vogel, "Acoustic control by wave field synthesis," vol. 93, no. 5, pp. 2764–2778, may 1993.

[78] Adam O'Donovan, Ramani Duraiswami, and Nail A. Gumerov, "Real time capture of audio images and their use with video," Oct. 2007, pp. 10–13.

[79] M. Monks, B.M. Oh, and J. Dorsey, "Audioptimization: Goal based acoustic design," *IEEE Computer Graphics & Applications*, pp. 76–91, May 2000.

[80] Patrick Cardinal, Pierre Dumouchel, Gilles Boulianne, and Michel Comeau, "GPU Accelerated Acoustic Likelihood Computations," in *Proc. of INTERSPEECH*, 2008.

[81] Paul R. Dixon and Tasuku Oonishia and Sadaoki Furuia, "Harnessing graphics processors for the fast computation of acoustic likelihoods in speech recognition," *Computer Speech & Language*, vol. 23, no. 4, pp. 510–526, Oct. 2009.

[82] Jike Chong, Youngmin Yi, Arlo Faria, Nadathur Rajagopalan Satish, and Kurt Keutzer, "Data-parallel large vocabulary continuous speech recognition on graphics processors," Tech. Rep. UCB/EECS-2008-69, EECS Department, University of California, Berkeley, May 2008.