Wave Field Synthesis for 3D Audio: Architectural Prospectives

Dimitris Theodoropoulos
dtheodor@ce.tudelft.nl

Cătălin Bogdan Ciobanu
C.B.Ciobanu@tudelft.nl

Georgi Kuzmanov
G.K.Kuzmanov@tudelft.nl

Computer Engineering Laboratory, Electrical Engineering Dept.
Delft University of Technology
Postbus 5031, 2600 GA, Delft
The Netherlands

ABSTRACT

In this paper, we compare the architectural perspectives of the Wave Field Synthesis (WFS) 3D-audio algorithm mapped on three different platforms: a General Purpose Processor (GPP), a Graphics Processor Unit (GPU) and a Field Programmable Gate Array (FPGA). Previous related work reveals that, up to now, WFS sound systems are based on standard PCs. However, on one hand, contemporary GPUs consist of many multiprocessors that can process data concurrently. On the other hand, recent FPGAs provide huge level of parallelism, and reasonably high performance potentials, which can be exploited very efficiently by smart designers. Furthermore, new parallel programming environments, such as the Compute Unified Device Architecture (CUDA) from NVidia and the Stream from ATI, give to the researchers full access to the GPU resources. We use the CUDA to map the WFS kernel on a GeForce 8600GT GPU. Additionally, we implement a reconfigurable and scalable hardware accelerator for the same kernel, and map it onto Virtex4 FPGAs. We compare both architectural approaches against a baseline GPP implementation on a Pentium D at 3.4 GHz. Our conclusion is that in highly demanding WFS-based audio systems, a low-cost GeForce 8600GT desktop GPU can achieve a speedup of up to 8x comparing to a modern Pentium D implementation. An FPGA-based WFS hardware accelerator consisting of a single rendering unit (RU), can provide a speedup of up 10x comparing to the Pentium D approach. It can fit into small FPGAs and consumes approximately 3 Watts. Furthermore, cascading multiple RUs into a larger FPGA, can boost processing throughput up to more than two orders of magnitude higher than a GPP-based implementation and an order of magnitude better than a low-cost GPU one.

Categories and Subject Descriptors

B.7.1 [**Hardware**]: [Algorithms implemented in hardware]

Keywords

3D Audio, Wave Field Synthesis, General Purpose GPU computing, Reconfigurable Computing

1. INTRODUCTION

Today, there are many multichannel audio technologies providing the so-called immersive or 3D-audio, which requires various speakers setups, consisting of up to hundreds of speakers. These sound reproduction techniques can be generally classified into three fundamentally different categories: 1) stereophony; 2) generation of the signals that actually reach the ears (binaural signals) [1]; and 3) synthesis of the wavefronts emitting from sound sources.

In this paper, we focus on the third category and more specifically on the Wave Field Synthesis (WFS) technology [2]. Related work reveals that all previous audio systems that utilize WFS technology, are based on standard PCs. Such an approach, however, introduces processing bottlenecks which limit the number of sound sources that can be rendered in real-time. Power consumption is also considerable because in most cases more than one PCs are required to drive a large number of speakers.

Meanwhile, over the last years, a new paradigm referred to as General Purpose Graphics Processor Unit (GPGPU) computing has gained researchers' attention. Many developers nowadays try to efficiently map various scientific problems [3], [4] (e.g. computational fluid dynamics, video games physics and digital signal processing) onto contemporary GPUs. This stems from the fact that the GPUs have evolved from specialized graphics engines to parallel processors with very high memory bandwidth, that can offer a GFLOPs per second performance of up to two orders of magnitude, comparing to a General Purpose Processor (GPP) [5].

Additionally, Field Programmable Gate Arrays (FPGAs) are well established on the field of accelerating various software applications, especially the ones that can be severely parallelized. Nowadays, FPGAs are utilized to accelerate a wide field of applications such as high definition multimedia [6], 3D-audio [7], bioinformatics [8], military applications [9] and many other categories.

Based on the fact that both GPUs and FPGAs have the potential of highly parallel data processing, we investigate under what conditions one or the other platform is performance and cost efficient for mapping the WFS 3D-audio algorithm (which can be considerably parallelized) onto them. The main contributions of this work are the following:

- We explore the architectural prospectives of the WFS algorithm by mapping it on three different platforms: a GPP, a GPU, and an FPGA
- We utilize NVidia's Compute Unified Device Architecture (CUDA) [5] parallel software environment to map the most computationally intensive parts of the WFS algorithm (WFS kernel) on a GeForce 8600GT. To the best of our knowledge, this is the first reported mapping and exploration of the WFS algorithm on a GPU.
- We quantified the performance gains from utilizing both GPU and FPGAs on real off-the-shelf chips. A small FPGA Virtex4 FX60 implementation of single Rendering Unit (RU) obtains WFS speedup of 10 times against a Pentium D GPP. A larger FPGA can accommodate up to 13 RUs, allowing a speedup of up to 140. Furthermore, we managed to achieve a speedup of up to eight times of the WFS kernel when mapped onto a low-cost GeForce 8600GT GPU.

The remainder of the paper is organized as follows: Section 2 discusses the WFS theoretical background and various audio systems based on this technology. In Section 3, we present the mapping process of the WFS kernel on the GPU. Section 4 briefly presents the WFS hardware accelerator already proposed in [7]. In Section 5, we compare all implementations, and Section 6 concludes the paper.

2. BACKGROUND AND RELATED WORK

In this section, we provide a theoretical background of the WFS algorithm. Furthermore, we discuss various audio systems that are based on the WFS technology.

One popular, yet computationally intensive, way of delivering a natural sound environment is provided by audio technologies that can synthesize wavefronts of a virtual source. The most important benefit of these technologies is that they do not constrain the listening area to a small surface, as it happens with stereophonic systems and binaural setups without headphones. On the contrary, wavefront synthesis provides a natural sound environment in the entire room, where every listener experiences an outstanding sound perception and localization. However, their main drawback is that they require large amount of data to be processed and many speakers to be driven.

The two technologies that synthesize wavefronts are Ambisonics and Wave Field Synthesis (WFS). Ambisonics was proposed from Oxford Mathematical Institute in 1970 [10]. Researchers focused on a new audio system that could recreate the original acoustic environment as convincingly as possible. In order to achieve this, they developed a recording technique that utilizes a special surround microphone, called the Soundfield microphone. The recording equipment generates 4-channel signal, called B-Format, that includes all the appropriate spacial information of the sound image. B-Format consists of left-right, front-back and up-down data, plus a pressure reference signal, providing the capability to deliver surround audio with height information. A major advantage of Ambisonics sound systems is that they can utilize an arbitrary number of loudspeakers that do not have to be placed rigidly.

WFS was proposed by Berkhout [2]. It is essentially based on Huygens' principle stating that a wavefront can be considered as a secondary source distribution. In the audio domain, Huygens' principle is applied by stating that a primary source wave front can be created by secondary audio sources (plane of speakers) that emit secondary wavefronts, the superposition of which creates the original one. However, some limitations arise in real world systems. For example, a plane of speakers is not feasible in practise, so a linear speaker array is used, which unavoidably introduces a finite distance between the speakers. This fact introduces artifacts such as spatial aliasing, truncation effects, amplitude and spectral errors of the emitted wavefront [11].

Theoretical background on WFS: Figure 1(a) illustrates an example of a linear array speaker setup. Each speaker has its own unique coordinates (xs_i, ys_i) inside the listening area. In order to drive each one of them so as the rendered sound source location is at A(x1, y1), the following operations are required to calculate the so called *Rayleigh 2.5D operator* [12]: filtering of the audio signals with a 3 dB/octave correction filter [13] and calculation of the delayed sample and its gain, according to each speaker distance from the virtual source. To render a source behind the speaker array, the inner product z between vector $\overrightarrow{d_1}$ and each speaker normal vector $\overrightarrow{n}$ must be calculated. Then the amplitude decay AD is given by the following formula [12]:

$$AD = \sqrt{\frac{Dz}{(Dz+z) * |\overrightarrow{d}|}} * cos(\theta) \tag{1}$$

where Dz is called *reference distance*, the distance where the Rayleigh 2.5D operator can give sources with correct amplitude, $|\overrightarrow{d}| = |\overrightarrow{d_1}|$, and $cos(\theta)$ is the cosine of angle θ between the vectors $\overrightarrow{d_1}$ and $\overrightarrow{n}$, as shown in Figure 1(a).

In order to render a moving source from a point A to a point B behind the speaker array, a linearly interpolated trajectory is calculated [12]: Distance $|\overrightarrow{d_2}|-|\overrightarrow{d_1}|$ is divided by the samples buffer size bs, in order to calculate the distance difference υ (in meters) of the source from speaker i between two consecutive audio samples:

$$\upsilon = \frac{|\overrightarrow{d_2}| - |\overrightarrow{d_1}|}{bs} \tag{2}$$

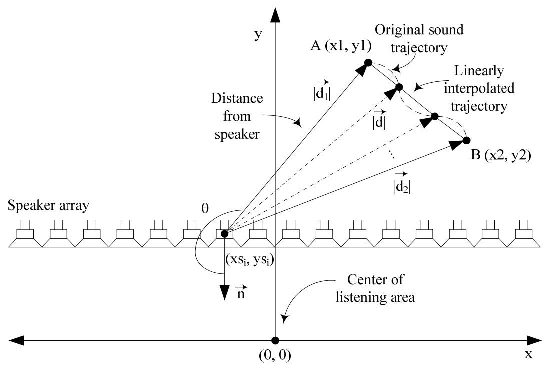
Based on the distance υ, the source distance $|\overrightarrow{d}|$ from speaker i with coordinates (xs_i, ys_i) is updated for every sample by the formula:

$$|\overrightarrow{d}| \leftarrow |\overrightarrow{d}| + \upsilon \tag{3}$$

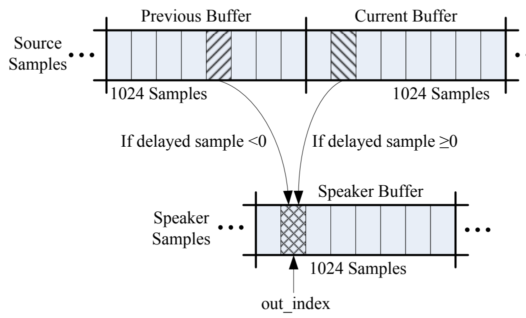
According to the current distance $|\overrightarrow{d}|$ from speaker i, an output sample is selected based on the formula:

$$delayedsample = -(l + (df * |\overrightarrow{d}|)) + (out\_index + +) \tag{4}$$

where $df = f_s/\upsilon_s$ is the distance factor (f_s is the sampling rate, υ_s is the sound speed), *out_index* is the current output audio sample and l is an artificial latency. Finally, the



(a) Speaker array setup.



(b) Proper choice of the delayed sample.

Figure 1: A hypothetical speaker array setup and an illustration of how the delayed sample is selected in WFS.

Buffer[delayed sample] is multiplied by the amplitude decay AD and the system master volume. The result is stored to the Speaker Buffer.

Figure 1(b) illustrates how the delayed sample is calculated. The source samples are divided into buffer-size source segments (in our implementation 1024-sample segments). In each iteration a source segment is used to select the proper audio samples for each speaker. However, there are cases where the evaluated delayed sample does not belong to the current source segment, but instead, to the previous one. Thus, in every iteration, two source segments are needed, the *current* and the *previous* one, to cover both cases where the evaluated delayed sample is positive or negative respectively. Further details can be found in [14], [15], [13] and [12].

Related work: A sound system that was built in IRT in Munich and called the Binaural Sky [16], actually combines both binaural and Wave Field Synthesis technologies. The Binaural Sky concept is based on the avoidance of Cross Talk Cancelation (CTS) filters real time calculation, while the listener's head is rotated. Instead of using two speakers the authors utilize a circular speaker array that synthesizes focused sound sources around the listener. The system uses a head tracking device and, instead of real time CTS filter calculation, it adjusts the speaker driving functions such as delay times and attenuations. The speaker array consists of 22 broadband speakers and a single low frequency driver. All real time processing is done on a Linux PC with a 22 channel sound card.

Two companies, SonicEmotion [17] and Iosono [18], produce audio systems based on WFS technology. SonicEmotion rendering unit is based on Intel Core2Duo processor and consumes an average power of 360 W. It supports rendering up to 64 real-time sound sources, while driving a 24 speaker array. Iosono rendering unit is also based on a standard PC approach and supports up to 64 real-time sources while driving 32 speakers. In both cases, when more speakers are required, additional rendering units have to be cascaded.

The authors of [19] describe a real-time immersive audio system that exploits WFS technology. The system performs sound recording from a remote location A, transmits it to another one B, and renders it through a speaker array utilizing the WFS technology. In order to preserve the original sound exact coordinates, a tracking device is employed. A beamformer also records the sound source, but without the acoustic properties of the recording location A. Thus, a dry source signal with its coordinates is transmitted to B. The WFS rendering unit receives this information along with the acoustic properties of the reproduction room B. The result is the same sound source being rendered exactly at the same position under B acoustic properties. The complete system consists of four PCs, out of which one used for the WFS rendering.

In [20], the authors developed a system that combines WFS technology with a projection-based multi-viewer stereo display. The system hardware setup consists of four standard PCs and 32 speakers. One PC is used to control two cameras that are tracking movements of a user. A second PC drives four LCD projectors that generate images on a perforated screen. A third PC is used as an audio player and is connected to Iosono rendering unit described above. The latter processes all input audio signals and drives 32 speakers divided into four 8-speaker panels.

Finally, one of the first attempts to utilize the GPU for 3D-audio algorithms processing was presented in [21]. The authors developed an optimized SSE assembly code running on a Pentium 4 at 3 GHz for binaural rendering. They also implemented an equivalent code to be run on an NVidia GeForce FX 5950 Ultra GPU, by using the Cg and OpenGL graphic languages. Their SSE implementation managed to render 700 real-time sound sources, while the GPU performed worse and rendered up to 580 sources. However, the authors identified that the GPU performance could be boosted by 50% if floating point texture re-sampling was supported in hardware. Furthermore, they suggested that the AGP connection could become a bottleneck when transferring large amounts of audio data from the GPU main memory to the audio hardware for playback. To the best of our knowledge, no HW implementation of WFS, neither on GPU, nor on FPGA has been proposed in the literature.

3. GPU BASED IMPLEMENTATION OF THE WFS KERNEL

In this section, we briefly describe the GPU organization and discuss the mapping details of the WFS kernel onto it.

GPU organization: Contemporary NVidia GPUs [5] consist of M multiprocessors, as illustrated in Figure 2, that can process data concurrently. Each multiprocessor includes P processors and the following four different types of on-chip memory:

1. A *shared memory* among all processors in a multiprocessor. The shared memory is used for data caching.

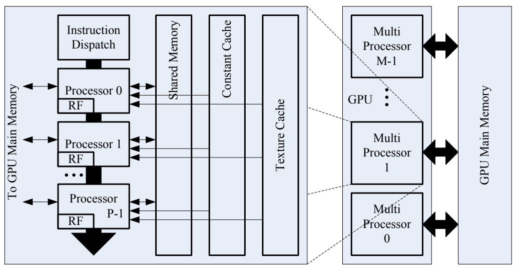


Figure 2: The GPU organization.

2. A read-only *constant memory* used for caching reads of constants from the GPU main memory.

3. Another read-only memory, called *texture memory*, that is used for caching textures read also from the GPU main memory.

4. P Register Files (RFs) distributed among all processors inside the multiprocessor.

In order for the developers to efficiently map general purpose kernels on the GPU without using specific graphics terms, such as textures, vertices and fragments, NVidia launched the CUDA parallel software environment [5]. CUDA introduces a set of extensions to the C programming language that exposes to the developers the parallel processing capabilities of the GPU. Each kernel mapped on the GPU is executed concurrently by many threads mapped on the multiprocessors. CUDA defines a thread hierarchy based on a grid of thread blocks. Each block consists of up to 512 threads in a 1-, 2- or 3-dimensional order, while the maximum dimension size of a grid of thread blocks can be up to 2^{16}.

Mapping of the WFS kernel onto the GPU: The WFS algorithm can be considerably parallelized. Once the source distance from a speaker and the source amplitude decay have been calculated, each sample for this particular speaker can be processed concurrently (eq. 4). Furthermore, all calculations regarding each speaker are also independent. Based on this fact, we decided to create a grid of thread blocks as depicted in Figure 3. The x-axis represents all thread blocks that process audio samples for a specific speaker. Thus, the total number of rows equals to the number of speakers. The y-axis represents all thread blocks that process a certain 1024-samples segment, which is the buffer size *bs* mentioned in Section 2. Thus, the number of columns is equal to $\frac{\#audiosamples}{buffersize}$. However, since each thread block can consist of up to 512, we split the source samples into two sets, {0,..., 511} and {512,...,1023}. Consequently, each thread is calculating serially two samples. Finally, the z-axis represents the total number of input sound sources that need to be rendered through the speakers.

While mapping the WFS kernel onto the GPU special attention was paid on the data transfers between the GPU main memory and the on-chip shared memory. As recommended in [5], while a GPU kernel is running, the GPU main memory accesses by the threads should be minimized, because each one requires hundreds of cycles. For this reason, source and speaker coordinates are stored in the constant memory, because it is cachable and thus will be faster to read them, instead of accessing the GPU main memory each time

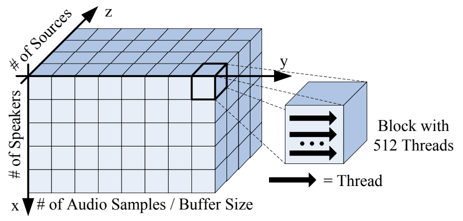


Figure 3: Grid of thread blocks for the WFS kernel.

they are required. When the WFS kernel starts execution, its first task is to load all required data from the GPU main memory to the shared memory. When data loading is done, all threads within a block are synchronized to make sure that further shared memory readings will be valid. Once all threads are synchronized, each one of them processes two speaker samples as mentioned earlier. All results are temporarily stored in the shared memory and, as soon as all threads are done, the kernel copies them back to the GPU main memory. Following this approach regarding the data transfers, we managed to minimize the memory access cost impact on the overall kernel execution time.

Overall, the processing steps of the WFS kernel for a source i rendered through speaker j can be summarized as follows:

1. *Copy from the GPU main memory the next 2048 source samples to the shared memory.* For every 1024 speaker samples to be calculated, the algorithm requires to access the *previous* source buffer when the delayed sample is negative, or the *current* source buffer when the delayed sample is positive (eq. 4).

2. *Calculate the amplitude decay and distance of source i from speaker j.* A single thread per block calculates the source i distance from speaker j and the corresponding amplitude decay. The reason that we use a single thread is because the distance and the amplitude are common for a specific 1024-sample segment.

3. *Process speaker samples 0 to 511.* All threads process one sample, and store the results back into the shared memory.

4. *Process speaker samples 512 to 1023.* Again each thread processes one sample and stores the results back into the shared memory.

5. *Copy processed speaker samples to the GPU main memory.* As soon as all speaker samples are processed, the threads are again synchronized and each one stores two speaker samples back to the main GPU memory.

Figure 4 depicts a flowchart of the complete software that runs both on the GPP and on the GPU. Once all required memory allocations are completed, the GPP filters all audio samples. The filtered audio samples are copied from the GPP main memory to the GPU main memory. The software then invokes the WFS kernel, which starts running on the GPU as explained before. When the WFS kernel finishes processing all filtered audio samples from all sources, the software copies all speakers samples back into the GPP main memory.

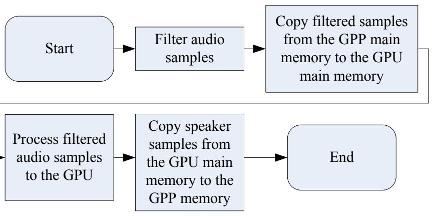

Start
Filter audio samples
Copy filtered samples from the GPP main memory to the GPU main memory
Process filtered audio samples to the GPU
Copy speaker samples from the GPU main memory to the GPP memory
End

Figure 4: Flowchart that illustrates the main WFS processing steps.

4. FPGA BASED IMPLEMENTATION OF THE WFS KERNEL

In this section, we briefly discuss the FPGA implementation of the WFS kernel, originally proposed in [7].

Complete infrastructure: Figure 5(a) illustrates the hardware design infrastructure. The PowerPC utilizes a 128-Kbytes instruction memory connected to the Processor Local Bus (PLB) through its PORTA. Furthermore, a 64-Mbytes DDR SDRAM is used to store audio samples. An additional 128-Kbytes data memory is also connected through its PORTA to the PLB that is used from the PowerPC. The PORTB is connected directly to the Fabric Coprocessor Module (FCM) [1] which accommodates the WFS hardware accelerator. This shared memory implementation allows the FCM to access the memory more efficiently compared to accessing it through the PLB. The FCM is also connected directly to the PowerPC through the Auxiliary Processor unit (APU) [22] interface. The latter is configured to decode one User Defined Instruction (UDI) that invokes the FCM execution. Finally, an RS232 module is employed to debug the system through the On-chip Peripheral Bus (OPB).

WFS kernel accelerator: For the internal calculations, the FCM does not follow a floating point approach. This would result in a complex hardware design capable of delivering results with unnecessary high accuracy. Instead, after experiments [7], it was decided that a fixed point format with five integer and 17 fractional bits to represent initial data, suffices to produce results with three decimal digits accuracy.

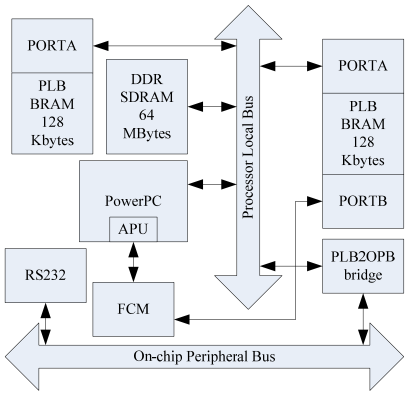
In Figure 5(b), the FCM internal organization is depicted. The hardware accelerator mainly consists of two different sub-units, a Rendering Unit (RU) and an FIR filter. Furthermore, a RU includes two modules, the *Preprocessor* and the *WFS Engine*, and a memory where all speaker coordinates are stored in the aforementioned fixed point format.

The Preprocessor is designed to calculate the samples distance υ (eq. 2), amplitude decay AD (eq. 1) and the source distance $\overrightarrow{d}$ (eq. 3) for each speaker. A more detailed operation analysis suggests that a total of nine additions/subtractions, nine multiplications, three square root operations and two divisions are required per speaker. The Preprocessor requires 142 clock cycles to complete data processing and the final results are forwarded to the WFS Engine.

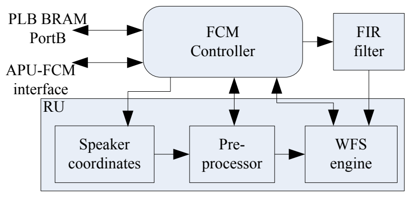
As soon as the Preprocessor has finished calculations, it acknowledges the FCM primary controller. The latter starts the WFS Engine which reads the amplitude decay, sample and source distance with respect to a particular speaker, and calculates the delayed samples (eq. 4). The WFS Engine integrates a reconfigurable number of Sample Selection Cores (SSCs), each one processing one speaker sample per cycle. Each SSC contains one multiplier, one subtractor, two accumulators and one adder. Thus, for example, a WFS with two SSCs can process two samples per cycle, which leads to 512 cycles for 1024 samples. In addition, 11 cycles are spent on communication among the WFS Engine internal modules, leading to a total of 523 cycles to process 1024 audio samples per speaker.

Figure 6 depicts the FCM functionality. As soon as a UDI is decoded from the PowerPC, the APU invokes the FCM execution. The latter reads the current source coordinates inside the listening area and the appropriate source samples from the PLB memory. The source samples are filtered and then stored to an internal buffer. The FCM controller loads the speakers coordinates from the memory and forwards them to the Preprocessor along with the source coordinates. The Preprocessor and the WFS Engine work in parallel; while the fist is calculating the amplitude decay and the sample and source distance for a particular speaker, the latter is calculating the samples of the previous speaker. With this pipelined module execution approach, the Preprocessor latency is completely hidden, thus improving considerably the FCM performance.

Table 1 displays the FPGA resource utilization with one RU with two SSCs integrated in the FCM. All slices occupied by infrastructural modules such as the PLB, OPB, PLB2OPB bridge and the RS232, are mentioned in the *Infrastructural slices* row. Table 2 shows how many such RUs can fit in a single FPGA, based on its available resources. Since one RU occupies only 3032 slices, the largest FPGA of the Virtex4 FX family can accommodate up to 13 RUs,



(a) Complete Design Infrastructure.



(b) FCM organization.

Figure 5: The complete WFS design infrastructure and the FCM organization.

[1]We follow Xilinx terminology.

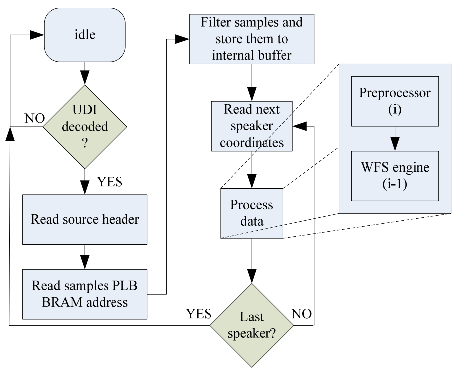

idle
NO
UDI decoded ?
YES
Read source header
Read samples PLB BRAM address
Filter samples and store them to internal buffer
Read next speaker coordinates
Process data
Preprocessor (i)
WFS engine (i-1)
YES
Last speaker?
NO

Figure 6: Flowchart that shows FCM functionality.

Table 1: FPGA resource utilization

Maximum frequency (MHz)	218
Total Power Consumption (W)	3
XtremeDSP Slices	14
RU Slices	3032
FIR Filter Slices	7152
Peripheral Slices	2205
Total Slices	12389

thus providing resources for considerable parallel processing of samples.

5. EXPERIMENTAL RESULTS

In this section, we provide the comparison among the three different WFS kernel implementations. First, we discuss the experimental setup of each platform and then we present the achieved speedup under various speakers configurations. Finally, we investigate how the data memory transfers overhead affects the overall WFS kernel execution.

Experimental setups: The original WFS kernel is implemented in the C software language and executed on a Pentium D D940 at 3.4 GHz with 2 GBytes of DDR2 SDRAM. The GPU modified WFS kernel implementation was mapped onto an NVidia GeForce 8600GT graphics card with 256 MBytes of DDR3 SDRAM. The 8600GT GPU consists of four multiprocessors, each one integrating 8 processors operating at 1.19 GHz [5]. Regarding the software environment, we installed the CUDA 2.1 and the latest NVidia drivers. Finally, the reconfigurable hardware implementation of the WFS kernel, was designed in VHDL. The complete design synthesis was done with Xilinx's EDK 9.1 and a fully functional prototype with one RU, including two SSCs, was implemented and tested on a Virtex4 FX60 -11.

Implementations comparison: Our primary goal is to compare the potential speedup that can be achieved by the different platform implementations, without taking into account the overhead of the *initial* data transfers between the GPP main memory and the GPU main memory. In other words, we assume that all data are already residing in the GPU main memory. Regarding the FPGA implementation, we assume that all samples are stored in on-chip BRAMs, thus excluding the overhead of transferring data between the FPGA and the DDR SRDAM. These assumptions are made based on the fact that, as we indicated in Section 1, in this work our goal is to identify the optimal architectural solutions for the WFS kernel *with respect to particular computational demands and in different application contexts.* Furthermore, we also provide an investigation regarding the memory transfers impact on the kernel execution both for the GPU and the FPGA.

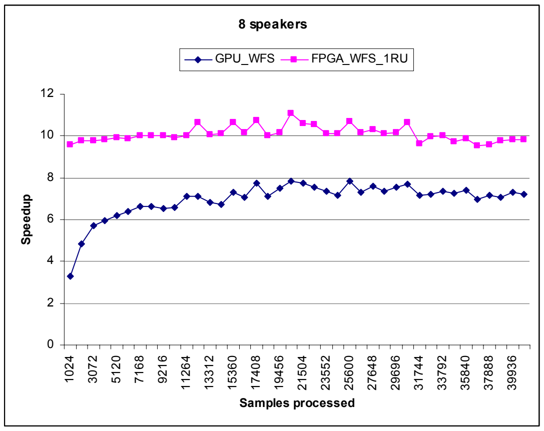
Figures 7(a), 7(b), 7(c) and 7(d) display the achieved speedup of the GPU and the FPGA implementation WFS kernel, called *GPU_WFS* and *FPGA_WFS_1RU* respectively, against the original implementation on the Pentium D. On the x-axis the number of processed source samples are projected, while the y-axis shows the achieved speedup. We investigated four different speaker setups ranging from eight up to 32 speakers. As one can observe, the GPU_WFS is up to 8x times faster than the Pentium D implementation, while the FPGA_WFS_1RU outperforms Pentium D 10x times. We believe that the latest generation of *inexpensive* GPUs that integrate up to 30 multiprocessors [5], can potentially achieve at least an order of magnitude speedup, because the parallel structure of the WFS algorithm, would efficiently utilize all of the additional processors. Furthermore, we found that the FPGA prototype with one RU integrating two SSCs, outperforms the WFS kernel running on the GeForce 8600GT GPU. Additionally, Figures 8(a), 8(b), 8(c) and 8(d) depict the potential speedup that can be achieved when mapping more such RUs on larger FPGAs. As one can observe on these figures, exploiting the potential parallelism of a larger FPGA by mapping more RUs, can provide a potential speedup of up to two orders of magnitude against the Pentium D implementation.

We compared also the GPU and the FPGA WFS kernel implementation against the commercial products from SonicEmotion [17] and Iosono [18]. According to their specifications[1], SonicEmotion's WFS rendering unit can support up to 64 sources when driving 24 speakers, while Iosono's rendering unit can also support 64 sources when driving 32 speakers. If the speaker setup consists of more than 24 and 32 speakers, then additional SonicEmotion and Iosono rendering units must be employed respectively. Figure 9 illustrates the comparison among all four platforms. As one can notice, both the GeForce GPU and the FPGA implementations, can support much higher number of real-time sources than the commercial PC-based products, even while using only a single GPU and a single FPGA. This fact leads to the conclusion that contemporary GPUs and FPGAs can greatly improve the performance of the WFS algorithm. Furthermore, the experimental results suggest that the performance of one GPU is comparable to a reconfigurable solution with one or two RUs, that can fit in a relatively small FPGA.

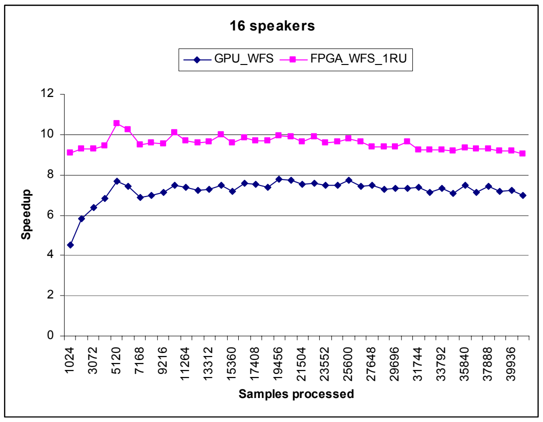
Table 2: Slices versus XtremeDSP slices proportion

FPGA	Available Slices	Slices / XtremeDSP	RUs Fit
V4FX40	9267	193	3
V4FX60	15923	124	5
V4FX100	32819	205	10
V4FX140	53811	280	13

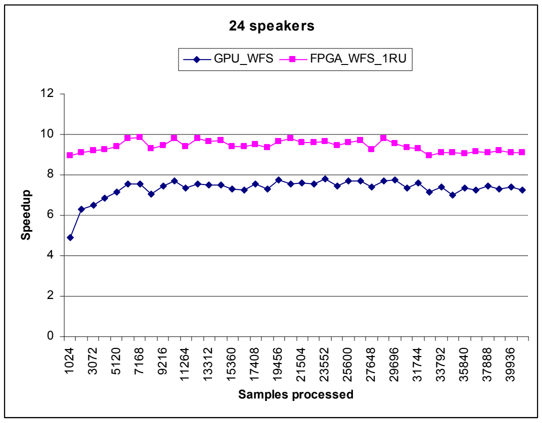
[1]SonicEmotion rendering-unit data were confirmed by personal communication with SonicEmotion at info@sonicemotion.com. In order to derive a performance estimation of Iosono's rendering unit, we considered [23] from the official Fraunhofer web site, stating that six Iosono PCs are used to drive 192 speakers.



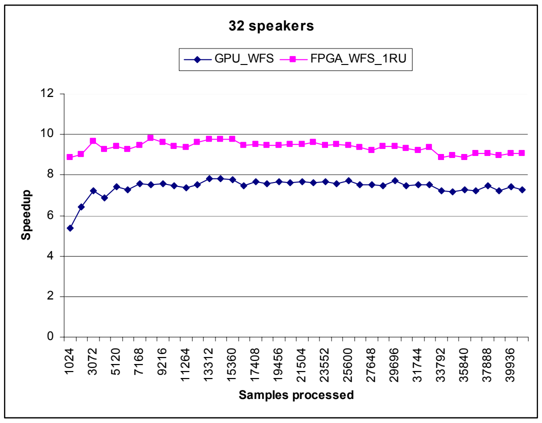
(a)



(b)



(c)



(d)

Figure 7: Speedup achieved against the GPP with 8, 16, 24 and 32 speakers.

When a large FPGA is considered, multiple RUs can be implemented which provides an order of magnitude higher performance of the FPGA against a low-cost GPU and almost two orders of magnitude against a modern GPP.

Finally, Table 3 generalizes the advantages and the disadvantages of each approach regarding performance, development time and power consumption. Regarding power efficiency, contemporary GPPs and GPUs require tens of Watts and, in some cases, up to more than 100 Watts. However, when the speaker setup consists of more than 32 speakers, still it is more power efficient to map the WFS kernel on the GPU comparing to a GPP approach. The reason for that is because one would require to employ additional rendering units to drive all the speakers, thus increasing the total consumed power, while a single GPU could keep up with the required processing demands. Furthermore, comparing the GPU and the current FPGA prototype, the latter consumes approximately 3 Watts [7], two orders of magnitude less, comparing to both the GPP and GPU. However, practical experience suggests that designing hardware modules with hardware description languages is a more cumbersome process than using high-level software ones. Thus, although the FPGA approach offers the best solution regarding performance and power consumption, it still requires a higher effort to build a complete hardware accelerator.

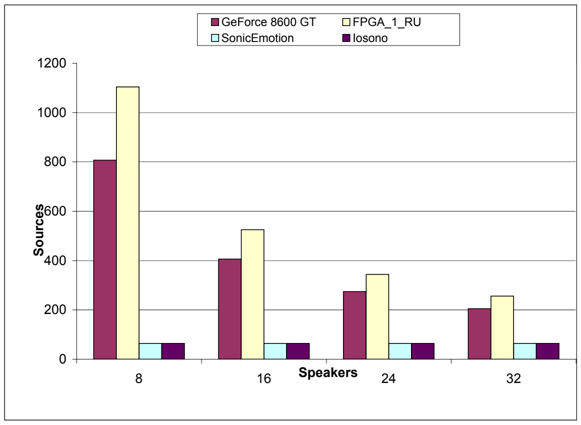
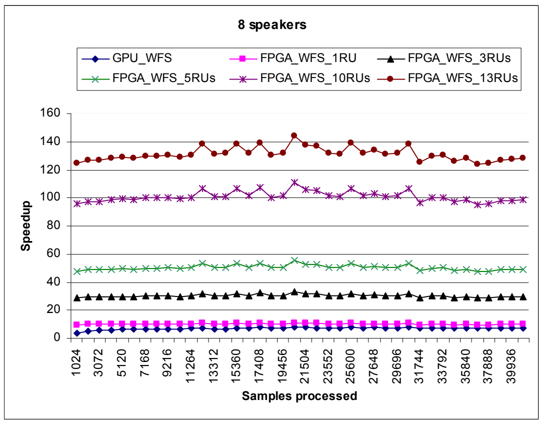
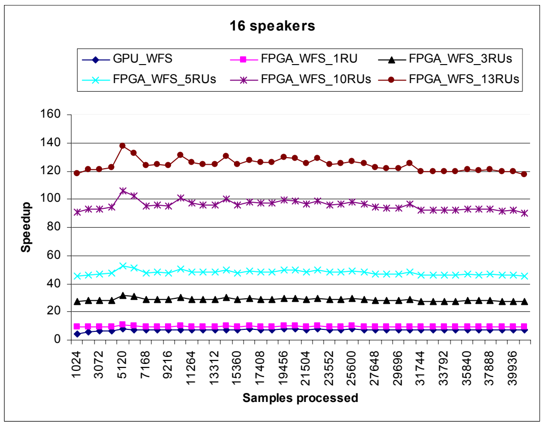


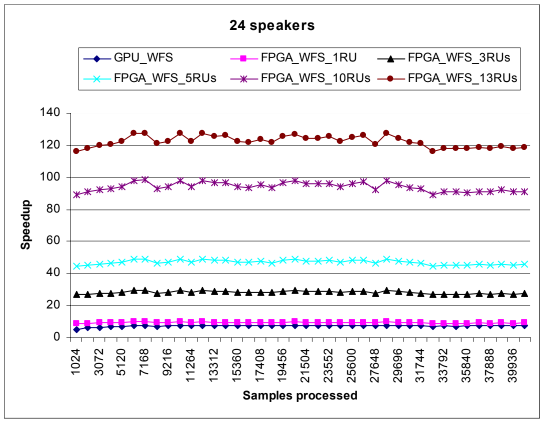
Figure 9: Number of real-time rendered sources that can be supported by each platform.



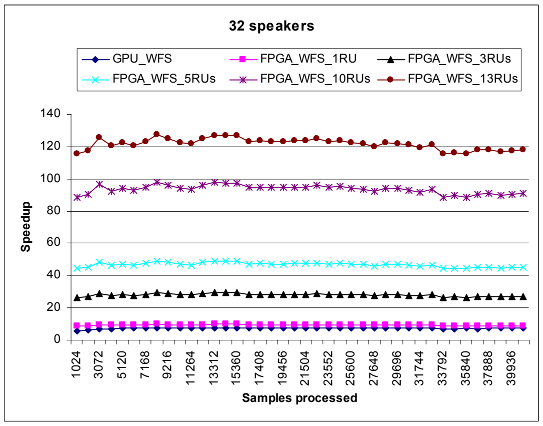
(a)



(b)



(c)



(d)

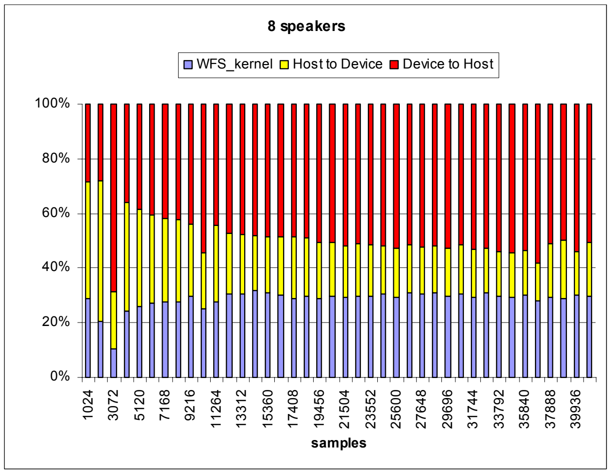
Figure 8: Speedup achieved against the GPP with 8, 16, 24 and 32 speakers using multiple RUs.

Table 3: Design features of WFS per implementation platform

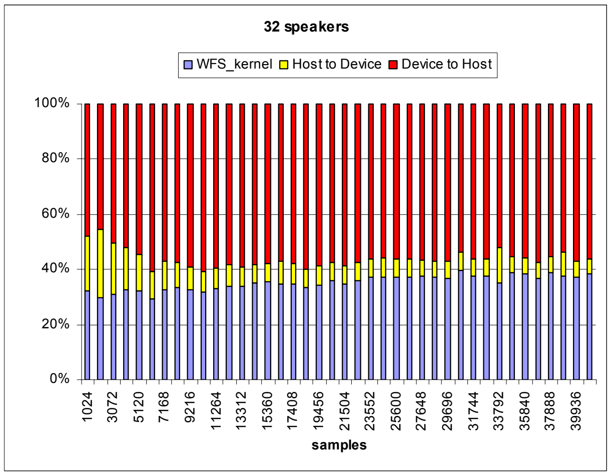
Platform	Performance	Development time	Power consumption
GPU	Very good	Medium	High
FPGA	Excellent	Long	Very low
GPP	Low	Short	High

Exploration of the memory transfers overhead on the PC: During our experiments with the GPU implementations, reported earlier in this study, we deliberately avoided to include the data memory transfers into the reported figures. The reason was that our goal was not to evaluate the memory subsystem performance of a PC, but to make an architectural study of the WFS kernel, assuming a dedicated subsystem with GPU being the main processor. Nevertheless, we explore the utilization of the GPU as part of the entire PC system, because such a platform provides the easiest and the most immediate way to use a GPU for a WFS-based audio system. Moreover, we can clearly indicate the memory transfers overhead, due to the PC memory / bus infrastructure and to use this knowledge for the design of a better dedicated memory subsystem.

For our study, we have divided the total WFS execution time on the GPU into three different parts: 1) *Host to Device* - the required time to transfer all data from the host (GPP) main memory to the device (GPU) main memory. 2) *WFS_kernel* - the actual WFS kernel execution time. 3) *Device to Host* - the transfer of the results from the device (GPU) main memory to the host (GPP) one. Figures 10(a) and 10(b) illustrate how these three different times are distributed among the total GPU execution time. As one can notice, in the case where we have 32 speakers, the *Device to Host* time occupies up to 60% of the total time. In contrast, in the case where we have only eight speakers to drive, this time occupies up to 40% of the total execution time. The reason for that is because in the first case, the WFS produces an increased amount of data to be transferred to the GPP memory, since there are more speakers to be driven. The *Host to Device* time for the initial data that have to be transferred from the GPP memory to the GPU one (such as source coordinates and filtered samples), is always the same, thus its portion is decreased as the number of speakers increases. Furthermore, the *WFS* time increases from roughly 30% when there are eight speakers, to 40% when there are 32 speakers. The conclusion from this analysis is that, as expected, data transfers overhead severely impacts the WFS execution time in the GPU with a typical memory hierarchy of the PC systems. This impact is quantified in Fig-

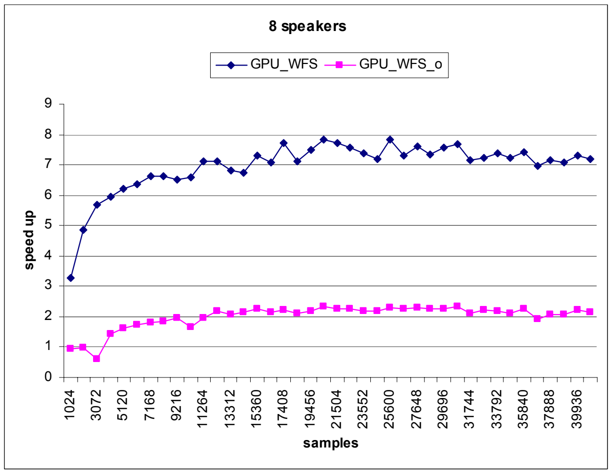


(a)

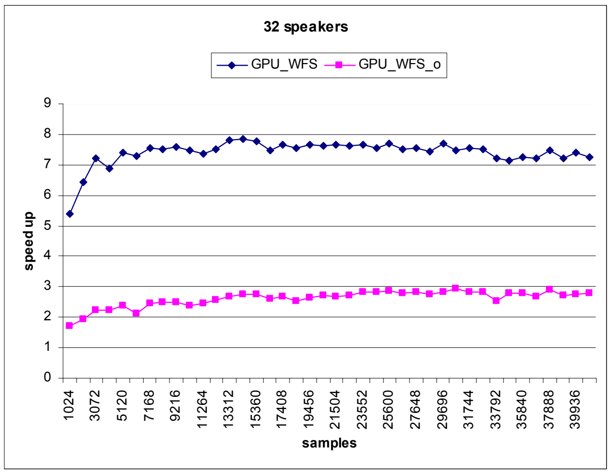


(b)

Figure 10: Breakdown of the GPU time execution with 8 and 32 speakers.



(a)



(b)

Figure 11: WFS kernel execution speedup with and without the memory overhead for 8 and 32 speakers.

ures 11(a) and 11(b), where *GPU_WFS* and *GPU_WFS_o* is the achieved speedup without and with the memory overhead respectively. The figures provide clear evidence, that even in the non optimal PC bus / memory organization a speedup of the WFS kernel can be still achieved using a GPU attached to the system.

As far as the FPGA implementation is concerned, the prospective system would have a dedicated memory subsystem by default. As we mentioned in Section 4, a RU with two SSCs process 1024 16-bits samples every 523 cycles at 200 MHz, thus approximately 772 MBytes/sec. In the Xilinx ML410 board, where the prototype was validated, there is a 32-bit word 64 MBytes DDR SDRAM chip running at 266 MHz. The memory bandwidth, therefore can be roughly up to $2 * 266 * 10^6 * 4$ bytes = 2.12 GBytes/sec. Consequently, a direct SDRAM connection with the FCM could alleviate, for example, the data transfer bottleneck and provide the required bandwidth. Recent technologies provide even higher operational frequencies for the contemporary DRAM modules. Therefore, our conclusion is that memory transfers should not introduce a performance bottleneck in WFS processing, neither on an FPGA, nor on a GPU, if the memory hierarchy is designed and optimized accordingly.

6. CONCLUSIONS

In this paper, we investigated the architectural perspectives of three different hardware platforms to map the WFS kernel: a GPP, a GPU, and an FPGA. We found that a GeForce 8600GT consisting of only four multiprocessors (latest inexpensive desktop GPUs consist of up to 30), can achieve up to 8x speedup comparing to the Pentium D 3.4 GHz implementation. The FPGA implementation achieves the best performance, even when utilizing only one RU with two SSCs. We also compared both approaches against the commercial products from Iosono and SonicEmotion, and found that both of them were outperformed from the GPU and FPGA implementations. The FPGA implementation is the one that can offer the best performance and lowest power consumption, outperforming the others by approximately two orders of magnitude. However, its main drawback is that it requires an increased effort to design the complete hardware accelerator. Our main conclusion is that a small FPGA (Virtex4 FX40) can provide a speedup in the same order as a single low-cost GeForce 8600GT GPU against a GPP imlementation. Due to the larger potential parallelism, that can be obtained on an FPGA compared to a low-cost GPU, a larger FPGA (e.g. Virtex4 FX140) can obtain up to an order of magnitude speedup compared to the GPU and up to two orders of magnitude, compared to a GPP. However, latest desktop GPUs consisting of up to 30 multiprocessors should be able to offer a boosted performance, due the parallel structure of the WFS algorithm that could exploit the additional processors efficiently. Furthermore, CUDA provides an efficient software environment that allows the annotation of the original C code with the required extensions, to exploit the potential parallelism offered from the latest GPUs. Ultimately, we found that, in order to achieve the aforementioned speedups, dedicated memory subsystem designs are required to saturate with the increased number of data transfers.

7. ACKNOWLEDGEMENTS

This work was partially sponsored by hArtes, a project (IST-035143) of the Sixth Framework Programme of the European Community under the thematic area "Embedded Systems"; and the Dutch Technology Foundation STW, applied science division of NWO and the Technology Program of the Dutch Ministry of Economic Affairs (project DCS.7533).

The authors would like to explicitly thank Lars Hörchens and Jasper van Dorp Schuitman from the Laboratory of Acoustical Imaging and Sound Control of the TU Delft for their valuable contribution to accomplish this work.

8. REFERENCES

[1] A. Mouchtaris, P. Reveliotis, and C. Kyriakakis, "Inverse of Filter Design for Immersive Audio Rendering Over Loudspeakers," in *IEEE Transactions on Multimedia*, vol. 2, June 2000, pp. 77–87.

[2] A. Berkhout, D. de Vries, and P. Vogel, "Acoustic Control by Wave Field Synthesis," in *Journal of the Acoustical Society of America*, vol. 93, May 1993, pp. 2764–2778.

[3] J. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU Computing," in *Proceedings of the IEEE*, vol. 96, May 2008, pp. 879–899.

[4] CUDA technology, NVidia Corporation, "http://www.nvidia.com/object/cuda_home.html#state=home."

[5] N. Corporation, "CUDA programming guide version 2.1," Tech. Rep., October 2008.

[6] Xilinx Inc., "H.264 Deblocker Core v1.0," May 2007.

[7] D. Theodoropoulos, G. Kuzmanov, and G. Gaydadjiev, "Reconfigurable Accelerator for WFS-Based 3D-Audio," in *IEEE Reconfigurable Architecture Workshop*, May 2009.

[8] Xcell Journal, "Using FPGA-Based Hybrid Computers for Bioinformatics Applications," October 2006, pp. 80–83.

[9] ——, "Future-Proofing Military Applications Using FPGAs," July 2007, pp. 21–23.

[10] M. A. Gerzon, "Periphony: With-Height Sound Reproduction," in *Journal of the Audio Engineering Society*, vol. 21, 1973, pp. 2–10.

[11] J. Daniel, R. Nicol, and S. Moreau, "Further Investigations of High Order Ambisonics and Wave Field Synthesis for Holophonic Sound Imaging," in *114th Convention of Audio Engineering Society*, March 2003, pp. 58–70.

[12] J. van Dorp Schuitman, "The Rayleigh 2.5D Operator Explained," Laboratory of Acoustical Imaging and Sound Control, TU Delft, The Netherlands, Tech. Rep., June 2007.

[13] P. Vogel, "Application of Wave Field Synthesis in Room Acoustics," Ph.D. dissertation, TU Delft, The Netherlands, 1993.

[14] M. Boone, E. Verheijen, and P. van Tol, "Spatial Sound Field Reproduction by Wave Field Synthesis," in *Journal of the Audio Engineering Society*, vol. 43, December 1995, pp. 1003–1012.

[15] W. P. J. D. Bruijn, "Application of Wave Field Synthesis in Videoconferencing," Ph.D. dissertation, TU Delft, The Netherlands, October 2004.

[16] D. Menzel, H. Wittek, G. Theile, and H. Fast, "The Binaural Sky: A Virtual Headphone for Binaural Room Synthesis," in *International Tonmeister Symposium*, October 2005.

[17] SonicEmotion Company, "http://www.sonicemotion.com."

[18] Iosono Company, "http://www.iosono-sound.com."

[19] H. Teutsch, S. Spors, W. Herbordt, W. Kellermann, and R. Rabenstein, "An Integrated Real-Time System For Immersive Audio Applications," in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, October 2003, pp. 67–70.

[20] J. P. Springer, C. Sladeczek, M. Scheffler, J. Hochstrate, F. Melchior, and B. Fröhlich, "Combining Wave Field Synthesis And Multi-viewer Stereo Displays," in *IEEE Virtual Reality Conference*, 2006, pp. 237–240.

[21] E. Gallo and N. Tsingos, "Efficient 3D Audio Processing on the GPU," in *Proceedings of the ACM Workshop on General Purpose Computing on Graphics Processors*, August 2004, p. C42.

[22] Xilinx Inc, "PowerPC 405 Processor Block Reference Guide," July 2005.

[23] Fraunhofer Institute for Digital Media Technology , "http://www.idmt.fraunhofer.de/eng/about_us/facts_figures.htm."