

# Spatial Sound for Video Games and Virtual Environments Utilizing Real-Time GPU-Based Convolution

Brent Cowan and Bill Kapralos

Faculty of Business and Information Technology,  
Health Education Technology Research Unit,  
University of Ontario Institute of Technology,  
Oshawa, Ontario, Canada. L1H 7K4.

brent.cowan@mycampus.uoit.ca    bill.kapralos@uoit.ca

## ABSTRACT

The generation of spatial audio is computationally very demanding and therefore, accurate spatial audio is typically overlooked in games and virtual environments applications thus leading to a decrease in both performance and the user's sense of presence or immersion. Driven by the gaming industry and the great emphasis placed on the visual sense, consumer computer graphics hardware (and the graphics processing unit in particular), has greatly advanced in recent years, even outperforming the computational capacity of CPUs. This has allowed for real-time, interactive realistic graphics-based applications on typical consumer-level PCs. Despite the many similarities between the fields of spatial audio and computer graphics, computer graphics and image synthesis in particular, has advanced far beyond spatial audio given the emphasis placed on the generation of believable visual cues over other perceptual cues including auditory. Given the widespread use and availability of computer graphics hardware as well as the similarities that exist between the fields of spatial audio and image synthesis, this work investigates the application of graphics processing units for the computationally efficient generation of spatial audio for dynamic and interactive games and virtual environments. Here we present a real-time GPU-based convolution method and illustrate its superior efficiency to conventional, software-based, time-domain convolution.

## Categories and Subject Descriptors

I.3.1 [Computer Graphics]: Hardware Architecture—*Graphics processors*

## General Terms

Performance

## Keywords

GPU, game audio, 3D sound, convolution, real-time.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*FuturePlay 2008*, November 3–5, 2008, Toronto, Ontario, Canada.  
Copyright 2008 ACM 0-89791-88-6/97/05 ...\$5.00.

## 1. INTRODUCTION

Given the importance of spatial hearing to humans, incorporating spatialized sound cues in realistic simulations such as games and immersive virtual environment applications seems obvious. In fact, doing so can be beneficial for a variety of reasons. Spatial sound cues can add a better sense of “presence” or “immersion”, they can compensate for poor visual cues (graphics), lead to improved object localization and, at the very least, add a “pleasing quality” to the simulation or game [2, 23, 24]. Despite the importance of spatial sound, it is often overlooked by the majority of video games and immersive virtual environments where historically, emphasis has been placed on the visual senses instead [5, 6].

Collectively, “the process of rendering audible, by physical or mathematical modeling, the sound field of a sound source in space, in such a way as to simulate the binaural listening experience at a given position in the modeled space” is known as *auralization* [15]. The goal of auralization is to recreate a particular listening environment, taking into account the acoustics of the environment and the characteristics of the listener (see [12] for a thorough review of virtual audio and auralization). Auralization is typically accomplished by determining the *binaural room impulse response* (BRIR). The BRIR represents the response of a particular acoustical environment to sound energy and captures the room acoustics for a particular sound source and listener configuration. Once obtained, the BRIR can be used to filter, typically through a convolution process, the desired anechoic sound. When this filtered sound is presented to a listener the original sound environment is recreated. The BRIR can be considered as the signature of the room response for a particular sound source and human receiver. Although interlinked, for simplicity and reasons of practicality, the room response and the response of the human receiver are commonly determined separately and combined via a post-processing operation to provide an approximation to the actual BRIR [15]. The response of the room is known as the *room impulse response* (RIR) and captures the reflection properties (reverberation), diffraction, refraction, sound attenuation and absorption properties of a particular room configuration (e.g., the environmental context of a listening room or the “room acoustics”). The response of the human receiver captures the direction dependent effects introduced by the listener due to the listener’s physical make-up (e.g., pinna, head, shoulders, neck, and torso) and is known as the *head related transfer function* (HRTF). HRTFs en-

compass various sound localization cues including interaural time differences (ITDs), interaural level differences (ILDs), and the changes in the spectral shape of the sound reaching a listener. The HRTF modifies the spectrum and timing of sound signals reaching each ear in a location-dependent manner [3]. Various techniques are available for determining (measuring, calculating) both the HRTF and the RIR however, a detailed discussion of these techniques is beyond the scope of this paper (see [12] for greater details). The output of the techniques used to determine the HRTF and the RIR is typically a transfer function which forms the basis of a filter that can be used to modulate source sound material (e.g., anechoic or synthesized sound) via a convolution operation. When the filtered sounds are presented to the listener, in the case of HRTFs, they create the impression of a sound source located at the corresponding HRTF measurement position while when considering the RIR, recreate a particular acoustic environment. However, as with BRIR processing, convolution is a computationally expensive operation especially when considering long filters associated with HRTFs and RIRs (filters with 512 coefficients are not uncommon) thus limiting their use to non-real-time applications. Fundamental to the generation of virtual audio is the convolution operation which is still primarily performed in software. Although programmable DSP cards are available which allow for hardware-based convolution thus greatly improving performance, currently these boards are very specialized and typically only available to product developers and not the general consumer [9]. In contrast to the consumer-grade audio cards currently available, dedicated computer graphics hardware is evolving at a tremendous pace.

In an attempt to reduce computational requirements, a number of initiatives have investigated simplifying the HRTFs and RIRs. With respect to the HRTF, dimensionality reduction techniques such as principal components analysis, locally linear embedding, and Isomap, have been used to map high-dimensional HRTF data to a lower dimensionality and thus ease the computational requirements [13, 14, 22, 25]. Despite the improvements with respect to computational requirements, even dimensionality reduced HRTFs are still not applicable for real-time applications and although the amount of reduction can be increased thus improving performance, reducing the dimensionality of HRTFs too much may lead to perceptual consequences that render them impractical. Further investigations must be conducted in order to gain greater insight. With respect to the RIR, it usually ignored altogether and approximated by simply including reverberation generated with artificial reverberation techniques instead. These techniques are not necessarily concerned with recreating the exact reflections of any sound waves in the environment. Rather, they artificially recreate reverberation by simply presenting the listener with delayed and attenuated versions of a sound source. Although these delays and attenuation factors do not necessarily reflect the physical properties of the environment being simulated, they are adjusted until a desirable effect is achieved. Given the interactive nature of video games and their need for real-time processing, when accounted for, reverberation effects in video games are typically handled using such techniques.

A graphics processing unit (GPU) is a dedicated graphics rendering device whose purpose is to provide a high performance, visually rich, interactive 3D experience by exploiting the inherent parallelism in the feed-forward graphics pipeline

[16]. In contrast to consumer-grade audio cards, GPUs have moved away from the traditional fixed-function 3D graphics pipeline toward a flexible general-purpose computational engine that can currently implement many parallel algorithms directly using graphics hardware resulting in tremendous computational speed-ups. Due to a number of reasons including the explosion of the consumer videogame market and advances in manufacturing technology, GPUs are, on a dollar-per-dollar basis, the most powerful computational hardware, providing “tremendous memory bandwidth and computational horsepower” [18]. GPUs are also becoming faster and more powerful very quickly, far exceeding Moore’s Law applied to traditional microprocessors [7]. In fact, instead of doubling every 18 months as with CPUs, GPU performance increases by a factor of five every 18 months or doubles every eight months [10]. In contrast to older GPUs which contained a fixed-function pipeline with output limited to 8-bit-per-channel color values, the current GPUs include fully programmable processing units which support vectorized floating point operations [18]. As a result, a number of high level languages have been introduced to allow for the control of vertex and pixel pipelines [4].

In order to take advantage of the power inherent in GPUs in addition to their relatively low cost, recently, a number of efforts have investigated the use of GPUs to a variety of non-computer graphics applications. Collectively, this effort is known as “general purpose computing on the GPU” (GPGPU) and given the flexibility of GPUs, has led to a number of GPU-based applications, outside the scope for which GPUs were originally designed for [18]. Examples include solving differential equations and general linear algebra problems [18], image processing [27], computer vision [8], fast Fourier transform, and audio processing. That being said, currently GPUs do not support integers and associated operations including bit-wise logical operations making them ill-suited for a operations requiring such features (e.g., cryptography). A thorough review including a detailed summary of GPGPU-based applications is provided by Owens *et al.* [18] and will therefore not be provided here.

Given the potential computational efficiencies that may be obtained with GPUs and their applicability to audio processing, we have begun an initiative to employ the GPU in the generation of spatial audio for games and virtual environments. The ultimate goal of the project is generate accurate spatial audio in real-time for inclusion in games and virtual environments. In this paper focus is placed on performing the convolution process, which is fundamental to the generation of spatial sound, using the GPU. In particular, we present a GPU-based convolution method that is capable of filtering a one-dimensional signal in real-time. A comparison with conventional, software-based convolution demonstrates the effectiveness of the developed method.

## 1.1 Paper Organization

The remainder of this paper is organized as follows. Section 2 provides background information regarding graphic processing units (GPUs) and their use in spatial audio-based applications. Section 3 provides implementation details regarding the GPU-based convolution method. Performance measures are provided in Section 4 where a comparison with conventional, software-based convolution is made. A discussion of the results is also provided. Concluding remarks and plans for future research are presented in Section 5.

## 2. BACKGROUND AND PREVIOUS WORK

In this section, several research efforts that have employed the GPU for audio-based processing will be described. However, prior to doing this, a brief overview of the GPU is provided.

### 2.1 Graphics Processing Units - Overview

In computer graphics, rendering is accomplished using a *graphics pipeline* architecture whereby rendering of objects to the display is accomplished in stages and each stage is implemented as a separate piece of hardware. The input to the pipeline is a list of vertices expressed in object space while the output is an image in the framebuffer. The stages of the pipeline and their operation are as follows [18]:

**Vertex Stage** i) Transformation of each (object space) vertex into screen space, ii) formation of triangles from the vertices, and iii) per-vertex lighting calculations.

**Rasterization Stage** i) Determination of the screen position covered by each of the triangles formed in the previous stage, and ii) interpolation of vertex parameters across the triangle.

**Fragment Stage** Calculation of the color for each fragment output in the previous stage. Often, the color values come from textures which are stored in texture memory. Here the appropriate texture address is generated and the corresponding value is fetched and used to compute the fragment color.

**Composition Stage** Pixel values are determined from the fragments.

In contrast to “traditional” *fixed-function pipeline* with “modern” (programmable) GPUs, both the vertex and fragment stages (also known as the *vertex shaders* and *pixel shaders* respectively or collectively as *programmable shaders*), are user programmable. Programs written to control the vertex stage are known as *vertex programs* while programs written to control the fragment stage are known as *fragment programs*. Initially, these programs were written in assembly language although higher level languages have been introduced and currently available (e.g., e.g., Microsoft’s *high level shading language*, *OpenGL shading language* [21], and Nvidia’s *Cg* [17]). Generally, the input to both of these programmable stages are four-element vectors where each element represents a 32-bit floating number. The vertex stage will output a limited number of 32-bit, four element vectors while the fragment stage will output a maximum of four floating point, four element vectors which typically represent color. The fragment stage is capable of fetching data from texture memory (e.g., *memory gather*) but cannot alter the address of its output which is determined before processing of the fragment begins (incapable of *memory scatter*). In contrast, within the vertex stage, the position of input vertices can be altered thus affecting where the image pixels will be drawn (e.g., the vertex stage supports both *memory gather* and *scatter*) [18].

### 2.2 Graphics Processing Units and Audio Processing

Audio-based ray tracing using the GPU was implemented by Jedrzejewski to compute the propagation of reflections

in highly occluded environments and allow for source and the listener to move throughout the simulation without the need for a long precomputation phase [11]. A comparison of the method implemented with a GPU and CPU demonstrated that the GPU-based implementation was much more computationally efficient (32 *ms* vs. 500 *ms* to trace a ray of order 10 on the GPU and CPU respectively). Röber *et al.* presented a (low-frequency) wave-based acoustical modeling method that made use of the GPU and in particular, fragment shaders, 3D textures, and the OpenGL framebuffer objects extension, in order to take advantage of the inherent parallelism of wave-based solutions to acoustical modeling [20]. Röber *et al.* [19] also describe a ray-based acoustical modeling method that employed the GPU to allow for real-time acoustical simulations. Their framework was designed along existing (computer graphics) GPU-based raytracing systems suitably modified to handle sound wave propagation. HRTF filtering was performed using fragment shaders available on the GPU. A frame-rate of up to 25 *fps* was achieved using a detailed model of a living room containing 1,500 polygons.

Gallo and Tsingos considered the application of GPUs to variable delay-line (delaying the signal of each sound source by the propagation time of the sound wave) and filtering (filtering the audio signal to simulate directivity functions, occlusions, and interaction with the medium of propagation). Variable delay line and filtering are two common spatial audio processing operations [9]. Delaying the signal involved resampling the signal at non-integer index values and was performed using the GPU using texture resampling. Filtering is performed using a four-band equalizer and implemented on the GPU using a four-component dot-product. Sound signals were stored as RGBA textures where each of the components held a band-pass copy of the original signal. Experimental results indicated a performance increase associated with GPU-based spatial audio when compared to optimized software implementations of a standard CPU. Despite the promising results, their work also showed that there are still a number of shortcomings that limit the efficient use of GPU processors for “mainstream” audio processing. In particular, long 1D textures cannot be accessed easily, and infinite impulse response filters (commonly used in audio processing) cannot be implemented efficiently.

Tsingos and Gascuel [26] developed an occlusion and diffraction method that utilizes computer graphics hardware to perform fast sound visibility calculations that can account for specular reflections (diffuse reflections are not considered), absorption and diffraction caused by partial occluders. Specular reflections are handled using an image source approach. Diffraction is approximated by computing the fraction of sound that is blocked by obstacles between the path from the sound source to the receiver by considering the amount of volume of the first Fresnel ellipsoid that is blocked by the occluders. A visibility factor is computed using computer graphics hardware. A rendering of all occluders from the receiver’s position is performed and a count of all pixels not in the background is taken (pixels that are “set” e.g., not in the background, correspond to occluders). Their approach handles a discrete set of frequency bands ranging from 31Hz to 8kHz and is primarily focused on sounds for animations. Although experimental results are not extensive, their approach is capable of computing a frequency dependent visibility factor that, unlike other ray-based ap-

```

uniform vec4 v0;
varying vec2 texCoord;
uniform sampler2D tex0;
const int MAX = 200;
uniform float f[MAX];

void main ()
{
    float x = 1.0/v0.x;
    float y = 1.0/v0.y;
    int L = MAX;
    float scale = 1.0/float(L);
    vec2 c;
    float oldY;
    float total = 0.0;
    int tempInt, calc;

    vec3 base = vec3(0.0, 0.0, 0.0);

    //Setup
    c = texCoord;
    c.x -= float(L)*x;
    if(c.x<0.0)
    {
        c.x = c.x+1.0;
        c.y = c.y-y;
    }
    oldY = c.y;

    for(int i=0; i<L; i++)
    {
        c.y = oldY+floor(c.x)*y; base.r = texture2D(tex0, c).r;
        total += ((base.r*256.0+base.g)-128.0)*f[i]; c.x += x;
    }

    //Convert back to bytes
    tempInt = int(total*256.0)+32768;
    calc = tempInt/256;
    base.r=float(calc)/256.0;
    base.g=float(tempInt-calc*256)/256.0;
    base.b=0.0;

    gl_FragColor = vec4(base,1.0);
}

```

**Figure 1: OpenGL Shading Language (OGSL) source code that implements one-dimensional convolution using the GPU.**

proaches, takes advantage of graphics hardware to perform this in an efficient manner. Although their approach is not completely real-time, it is “capable of achieving interactive computation rates for fully dynamic complex environments” [26].

### 3. IMPLEMENTATION DETAILS

In this section, implementation details of the GPU-based convolution method are provided. The implementation is based on the OpenGL shading language (OGSL) [21]. It is assumed that the input (un-processed) signal is a short integer (e.g., 16-bit resolution), which is common amongst WAV files. Of course, given an input signal that does not conform to this assumption, it can still be processed but at additional computational cost. The filter considered here is an actual HRTF filter taken from the CIPIC HRTF dataset [1] (measured from a KEMAR manikin using the “large ear” with the sound source positioned on the horizontal axis directly in front of the KEMAR). Furthermore, it is assumed the filter coefficients are floating point numbers (e.g., “float”) and of size 200 (again, these assumptions can easily be relaxed). The OGSL source code that implements GPU-based convolution is provided in Figure 1.

GPUs have been designed to work with two-dimensional images as the output of typical computer graphics applications is a two-dimensional image. Therefore, prior to performing the convolution, there is a set-up phase to convert the one dimensional audio signal and filter, into a two-dimensional format as required by the GPU (this is not included in the source code listing in Figure 1). Many video

```

void TextureBW::Convert()
{
    GLubyte* tempData = new GLubyte[width*height*3];
    for(GLuint i=0; i<width*height; i++)
    {
        tempData[i*3] = (data[i]+32768)/256;
        tempData[i*3+1] = (data[i]+32768)%256;
        tempData[i*3+2] = 0;
    } //End loop
    ...
}

```

**Figure 2: Source code to convert the one-dimensional input signal into a two-dimensional format required by the GPU.**

```

...
glReadBuffer(GL_BACK);
glReadPixels(520, 0, tex3.width, tex3.height, GL_RED, GL_BYTE, Red);
glReadPixels(520, 0, tex3.width, tex3.height, GL_GREEN, GL_BYTE, Green);
for(GLuint i=0; i<size; i++)
{
    Output[i] = Red[i]*256 + Green[i];
}
...

```

**Figure 3: Source code to return the two-dimensional data back from the video card back into a one-dimensional format.**

cards do not allow intensity maps of 16 bits. For robustness the input signal is stored (and therefore must be converted) as an RGB image with 1 byte per color value. The 2 byte short integers are stored in the red and green portion of the image with the blue element left unused (see Figure 2). The image is  $256 \times 256$  pixels in size. Since the sound file sizes we tested do not completely fill the image, we render only the portion of the image containing sound data.

The function then goes on to create a texture from the data in OpenGL. This however is accomplished using the CPU and not the GPU. However, given that this step is required in order to ensure the data is in the required GPU format, as will be described in Section 4, the computational running time associated with this step is still included in the total computational requirements of the GPU-based convolution implementation. To return the data back from the video card, the video card output must be copied from the display (screen) into arrays of bytes. These byte arrays must then be combined to form the desired output (see Figure 3).

#### 3.1 Limitations

The majority of the currently available video cards support 8-bit channels only (although some, such as the NVIDIA 8800 GTX used in this work do support 16-bit channels). Therefore, although the shader could be written more efficiently making use of 16-bit channels, to ensure the proposed shader is able to be used with a typical graphics card, we assume 8-bit channel support only thus, each 16-bit sound sample was divided between the red and green channels, leaving the blue channel unused. This leads to two inefficiencies: i) computational time required for the conversion, and ii) unused (wasted) channel.

### 4. RESULTS AND DISCUSSION

In this section, the effectiveness of the GPU-based convolution method is presented. This is accomplished by i) comparing the computational requirements (running time) of the GPU-based convolution method and conventional convolution method implemented using the CPU (e.g., software-based), and ii) providing a graphical example of the convo-

Size	CPU Time (ms)	GPU Time (ms)
5000	7.52	3.69
10000	15.11	3.75
15000	22.70	3.80
20000	30.31	3.86
25000	37.91	3.94
30000	45.50	4.00
35000	53.09	4.06
40000	60.69	4.13
45000	68.28	4.17
50000	75.88	4.24
55000	83.47	4.28
60000	91.06	4.34

**Table 1: Computational time requirements.** The first column represents the size (number of samples) of the input signal, the second column represents the average (1000 iterations) computational time requirements of the conventional CPU-based (software) convolution, and the third column represents the average (1000 iterations) computational time requirements GPU-based (hardware) convolution.

lution process.

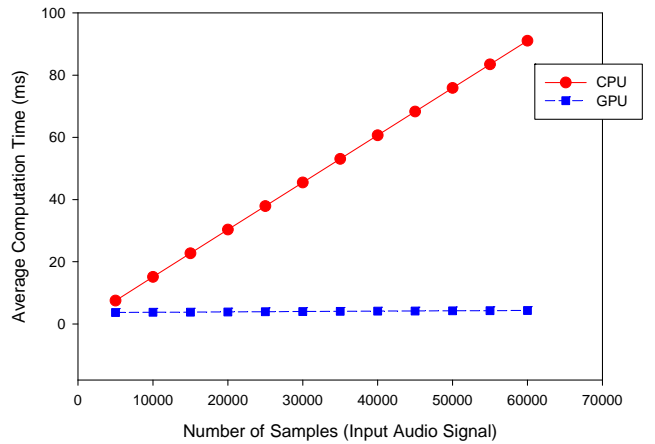
#### 4.1 Comparison

Here, a comparison of the computational running time requirements for both the conventional (software-based) and GPU-based convolution methods is made. This is accomplished by measuring the computational time requirements when convolving a particular input signal with an HRTF for each method (the same input signal and HRTF was used for both methods). As previously described, the filter was an actual HRTF filter (one-dimensional) from the CIPIC HRTF dataset [1] and contained 200 (floating point) coefficients. The input signal was a one-dimensional sine-wave signal (each sample had a resolution of 16 bits and of type “short int”). That being said, for the purposes of this test, what the input signal and filter actually represent is insignificant. The size (number of samples) of the input signal ranged from 5,000 to 60,000, increasing in increments of 5,000. The tests were performed using a Dell XPS 720 PC with an Intel Core2 6700 (2.66GHz) Processor with 2GB of RAM and an NVIDIA GeForce 8800 GTX graphics card which of course includes a programmable GPU.

The results of this test are illustrated in Figure 4, where the computational time requirements (x-axis) vs. the size of the input signal of conventional CPU-based (software) convolution and GPU-based (hardware) convolution are shown. Each point on the graph (both GPU and CPU-based implementations) represents computational time requirements averaged over 1,000 iterations. Furthermore, the computational time for the GPU includes processing on the CPU which was performed to convert the data into the format required by the GPU.

#### 4.2 Graphical Example

A graphical representation of the convolution process is provided in Figures 5 and 6. In particular, Figure 5(a) illustrates the higher order bytes of the original (one-dimensional) sine-wave signal after it has been converted into an “image format” to be loaded into the GPU. Figure 5(b) illustrates

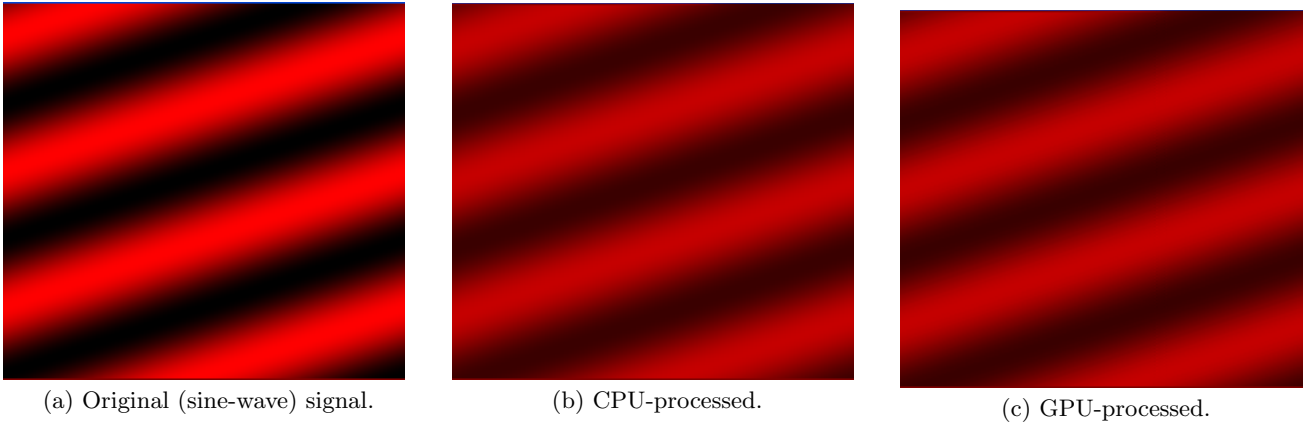


**Figure 4: A comparison of the computational time requirements (x-axis) vs. the size of the input signal of conventional CPU-based (software) convolution and GPU-based (hardware) convolution.**

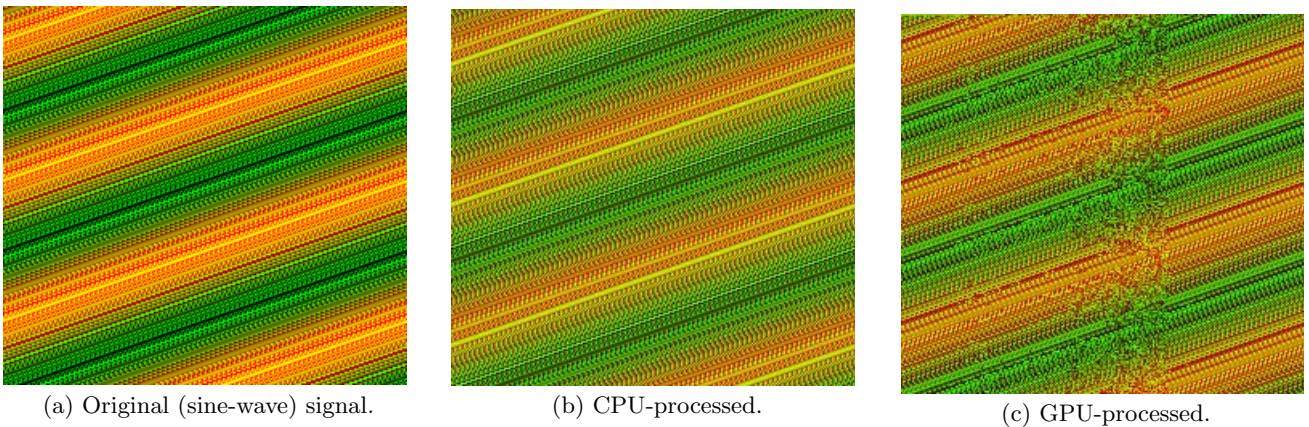
the higher order bytes of the filtered signal after convolving the original signal with the HRTF using conventional (software-based) convolution, while Figure 5(c) illustrates the higher order bytes of the filtered signal after convolving the original signal with the HRTF using the GPU-based convolution method. Visually, the two resulting signals appear identical without the introduction of any artifacts. The corresponding lower order bytes are illustrated in Figure 6. In contrast to the resulting higher order bytes of the GPU-processed signal, the addition of artifacts (noise) is apparent when examining the lower order bytes of the GPU-based filtered signal. These artifact (noise) are probably due to rounding errors.

#### 4.3 Discussion

Examination of Figure 4 clearly illustrates the superiority (with respect to computational time) of the GPU-based implementation. In particular, the GPU-based convolution method requires approximately 4.02 ms to compute irrespective of the input size, meeting real-time requirements. In contrast, the CPU computational time requirements increased linearly with the size of the input signal, starting at 7.52 ms for an input signal comprised of 5,000 samples and ending with 91.06 ms for an input signal comprised of 60,000 samples. A computational time of 4.02 ms indicates that the convolution of an input signal containing 60,000 sample with an HRTF (filter) containing 200 coefficients can be performed in real-time, corresponding to approximately 249 *fps* in contrast to the approximately 11 *fps* when considering software-based convolution. Furthermore, the size of the HRTF can be further reduced using dimensionality reduction techniques for example, to further reduce computational requirements [13, 14, 22, 25]. That being said, although HRTF processing is important to the generation of spatial sound, further processing is required to account for the acoustics of the environment (e.g., reverberation) which, depending on the approach taken, may substantially decrease the frame rate. In addition, despite the tremendous computational time savings, further investigation into the perceptual consequences of GPU-based HRTF process-



**Figure 5: Output sample - higher order bytes.** (a) Original (one-dimensional) sine-wave signal after it has been converted into an “image format” to be loaded into the GPU. (b) Filtered signal after convolving the original signal with the HRTF using conventional (software-based) convolution. (c) Filtered signal after convolving the original signal with the HRTF using the GPU-based convolution method.



**Figure 6: Output sample - lower order bytes.** (a) Original (one-dimensional) sine-wave signal after it has been converted into an “image format” to be loaded into the GPU. (b) Filtered signal after convolving the original signal with the HRTF using conventional (software-based) convolution. (c) Filtered signal after convolving the original signal with the HRTF using the GPU-based convolution method.

ing must be performed. In particular, as illustrated in Figure 6(c), noise/artifacts are introduced to the lower-order bytes of the resulting GPU-based convolution output. Finally, it should be noted that the PC used in this experiment is a higher-end gaming desktop PC with high-end graphics video card. Although the majority of computer users and even “gamers” may not necessarily have access to such a PC, its cost is drastically falling (currently, an XPS 720 PC with identical specifications to the PC used in these experiments is available for under \$2,000.00 (CAN)) and its technology will soon be available as standard on most PCs.

## 5. SUMMARY

In this paper we have presented a GPU-based convolution implementation using the OpenGL shading language (OGLS). Results indicate that the method is far more computationally efficient when compared to conventional, time-domain, software-based convolution and is in fact capable of performing convolution of a filter containing 200 coeffi-

cients and a one-dimensional signal of up to 60,000 samples, in real-time (approximately four milliseconds regardless the size of the one-dimensional input signal). Given that the generation of virtual audio hinges on the convolution operation and the widespread availability of computer graphics cards with onboard programmable GPUs, the generation of accurate virtual audio for games and virtual environments is now plausible.

## 5.1 Future Work

Future work will include human user studies to investigate whether there are any perceptual consequences when the GPU-based convolution method is used to filter an audio signal with HRTFs. Perceptual consequences may arise given the rounding errors that may be encountered when converting the input data (audio signal and HRTF filters) to a format required by the video card. Future work will also include the investigation of the use of the GPU to model the room impulse response of a particular environment and to

model occlusion and diffraction effects in an efficient manner.

## Acknowledgments

The financial support of the Natural Sciences and Engineering Research Council of Canada (NSERC) in the form of an Undergraduate Summer Research Award to Brent Cowan and a Discovery Grant to Bill Kapralos is gratefully acknowledged. The authors also thank the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis, for making their HRTF dataset freely available.

## 6. REFERENCES

- [1] V. R. Algazi, R. O. Duda, D. M. Thompson, and C. Avendano. The CIPIC HRTF database. In *2001 IEEE Workshop on Applications of Signal Processing to Acoustics*, pages 111–123, New Paltz, NY, USA, October 21–24 2001.
- [2] D. B. Anderson and M. A. Casey. The sound dimension. *IEEE Spectrum*, pages 46–50, March 1997.
- [3] R. Begault. *3-D Sound for Virtual Reality and Multimedia*. Academic Press, MA, USA, 1994.
- [4] I. Buck, T. Foley, D. Horn, J. Sugerma, K. Fatahalian, M. Houston, and P. Hanrahan. Brook for GPUs: Stream computing on graphics hardware. *ACM Transactions on Graphics*, 23(3):777–786, 2004.
- [5] S. Carlile. *Virtual Auditory Space: Generation and Application*. R. G. Landes Company, Austin, TX, USA, 1996.
- [6] M. Cohen and E. Wenzel. The design of multidimensional sound interfaces. In W. Barfield and T. Furness, editors, *Virtual Environments and Advanced Interface Design*, chapter 8, pages 291–346. Oxford University Press Inc., New York, NY, USA, 1995.
- [7] M. Ekman, F. Warg, and J. Nilsson. An in-depth look at computer performance growth. *Computer Architecture News*, 33(1):144–147, 1994.
- [8] J. Fung, F. Tang, and S. Mann. Mediated reality using computer graphics hardware for computer vision. In *Proceedings of the 6<sup>th</sup> IEEE International Symposium on Wearable Computers, 2002 (ISWC 2002)*, pages 83–89, Seattle, WA, USA, October 7–10 2002.
- [9] E. Gallos and N. Tsingos. Efficient 3D-audio processing with the GPU. In *Proceedings of the ACM Workshop on General Purpose Computing on Graphics Processors*, Los Angeles, CA, USA, August 7–8 2004.
- [10] D. Geer. Taking the graphics processor beyond graphics. *IEEE Computer*, pages 14–16, September 2005.
- [11] M. Jedrzejewski. Computation of room acoustics on programmable video hardware. Master’s thesis, Polish-Japanese Institute of Information Technology, Warsaw, Poland, 2004.
- [12] B. Kapralos, M. Jenkin, and E. Milios. Virtual audio systems. *Presence: Teleoperators and Virtual Environments*, 2008. To appear.
- [13] B. Kapralos and N. Mekuz. Application of dimensionality reduction techniques to HRTFs for interactive virtual environments. In *Proceedings of the ACM Advancements in Computer Entertainment (ACE 2007)*, Salzburg, Austria, June 13–15 2007.
- [14] D. J. Kistler and F. L. Wightman. A model of head-related transfer functions based on principle components analysis and minimum phase reconstruction. *Journal of the Acoustical Society of America*, 91(3):1637–1647, 1992.
- [15] M. Kleiner, D. I. Dalenback, and P. Svensson. Auralization - an overview. *Journal of the Audio Engineering Society*, 41(11):861–875, 1993.
- [16] D. Luebke and G. Humphreys. How GPUs work. *IEEE Computer*, pages 96–100, February 2007.
- [17] W. R. Mark, P. S. Glanville, K. Akeley, and M. J. Kilgard. Cg: a system for programming graphics hardware in a C-like language. In *Proceedings of the ACM International Conference on Computer Graphics and Interactive Techniques SIGGRAPH 2003*, pages 896–907, San Diego, CA, USA, July 27–31 2003.
- [18] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):80–113, 2007.
- [19] N. Rber, U. Kaminski, and M. Masuch. Ray acoustics using computer graphics technology. In *Proceedings of the 10<sup>th</sup> International Conference on Digital Audio Effects*, Bordeaux, France, September 10–15 2007.
- [20] N. Rober, M. Spindler, and M. Masuch. Waveguide-based room acoustics through graphics hardware. In *Proceedings of the International Computer Music Conference 2006*, New Orleans, LA, USA, November 6–11 2006.
- [21] R. Rost. *OpenGL Shading Language*. Addison-Wesley Professional, Boston, MA, USA, second edition, 2006.
- [22] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.
- [23] R. D. Shilling and B. Shinn-Cunningham. Virtual auditory displays. In K. Stanney, editor, *Handbook of Virtual Environment Technology*, pages 65–92. Lawrence Erlbaum Associates, Mahwah, NJ, USA, 2002.
- [24] U. P. Svensson and U. R. Kristiansen. Computational modeling and simulation of acoustic spaces. In *Proceedings of the 22<sup>nd</sup> International Conference on Virtual, Synthetic and Entertainment Audio*, pages 11–30, Espoo, Finland, 2002.
- [25] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.
- [26] N. Tsingos and J. D. Gascuel. Soundtracks for computer animation: Sound rendering in dynamic environments with occlusion. In *Proceedings of Graphics Interface ’97*, pages 9–16, Kelowna, BC, Canada., May 21–23, 1997.
- [27] R. Yang and G. Welch. Fast image segmentation and smoothing using commodity graphics hardware. *Journal of Graphics Tools*, 7(4):91–100, 2003.