GPU-Based One-Dimensional Convolution for Real-Time Spatial Sound Generation

Brent Cowan and Bill Kapralos
Game Development and Entrepreneurship Program, Faculty of Business and IT.
University of Ontario Institute of Technology.
2000 Simcoe Street North, Oshawa, Ontario, Canada. L1H 7K4.
brent.cowan@mycampus.uoit.ca bill.kapralos@uoit.ca

Abstract

Incorporating spatialized (3D) sound cues in dynamic and interactive videogames and immersive virtual environment applications is beneficial for a number of reasons, ultimately leading to an increase in presence and immersion. Despite the benefits of spatial sound cues, they are often overlooked in videogames and virtual environments where typically, emphasis is placed on the visual cues. Fundamental to the generation of spatial sound is the one-dimensional convolution operation which is computationally expensive, not lending itself to such real-time, dynamic applications. Driven by the gaming industry and the great emphasis placed on the visual sense, consumer computer graphics hardware, and the graphics processing unit (GPU) 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. Given the widespread use and availability of computer graphics hardware and the similarities that exist between the fields of spatial audio and image synthesis, here we describe the development of a GPU-based, one-dimensional convolution algorithm whose efficiency is superior to the conventional CPU-based convolution method. The primary purpose of the developed GPU-based convolution method is the computationally efficient generation of real-time spatial audio for dynamic and interactive videogames and virtual environments.

Author Keywords

Spatial sound; game audio; graphics processing unit; GPU; convolution; real-time.

Introduction

A virtual (or three-dimensional (3D), or spatial) audio system (or audio display) allows a listener to perceive the position of a sound source(s), emanating from a static number of stationary loudspeakers or a pair of headphones, as coming from arbitrary locations in three-dimensional space. Spatial sound technologies go far beyond traditional stereo and surround sound techniques by allowing a virtual sound source to have such attributes as left-right, back-forth, and up-down (Cohen and Wenzel, 1995). Incorporating spatialized auditory information in an immersive virtual environment and videogames is beneficial for a variety of reasons. Spatial auditory cues can add a better sense of "presence" or "immersion", compensate for poor visual cues (graphics), and at the very least, add a "pleasing quality" to the simulation (Durlach and A. S. Mavor, 1994; Shinn-Cunningham, 2002). Despite these benefits and despite the fact that spatial sound is a critical cue to the perception of our environment, it is often overlooked in immersive virtual environments and videogames where historically, emphasis has been placed on the visual senses

(Carlile, 1996; Cohen and Wenzel, 1995). That being said, the generation of spatial sound for dynamic, and interactive applications using traditional software-based methods and techniques is computationally very expensive except for trivial environments which are typically of little use.

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* (Kleiner *et al*., 1993). 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 Kapralos *et al*. (2008) for a thorough review of spatial 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 the desired anechoic sound through a convolution process described by Equation 1.

$$s[n] * r[n] = \sum_{k=-N/2+1}^{N/2} s[n-k]r[k] \quad \Leftrightarrow \quad S(f)R(f), \qquad (1)$$

Where s is the input signal, r is the filter, n denotes the sample index, S and R denote the discrete Fourier transform (DFT) of s and r, respectively, f denotes the DFT index and N denotes the filter sample size.

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 (Kleiner *et al*., 1993). 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 (i.e., 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 (i.e., pinna, head, shoulders, neck, and torso) and is known as the *Head Related Transfer Function* (HRTF). HRTFs encompass 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. 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 Kapralos *et al*. (2008) 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 and when considering the RIR, recreate a particular acoustic environment. However, as previously described, 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 (the operation described in Equation 1 must be performed for each input signal sample). Convolution is primarily performed in software in the time domain.

As shown in Equation 1, convolution in the time domain is equivalent to multiplication in the frequency domain and therefore, performance improvements can be made by performing the convolution operation in the frequency domain (Gardner, 1995). However, in order to accomplish this, the input and filters must be converted to their frequency domain representation using the *Fast Fourier Transform* (FFT); a time consuming process when performed using software thus also making it impractical for real-time use. Recent work in image processing has established a GPU-based convolution method capable of performing a two-dimensional convolution operation in real-time (Fialka and Cadik, 2006). In addition to software-based convolution methods, programmable DSP cards are available which allow for hardware-based convolution thus greatly improving performance. However, these boards are very specialized and typically only available to product developers and not the general consumer (Gallos and Tsingos, 2003). 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 (Kapralos and Mekuz, 2007; Kistler and Wightman, 1992). 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 is 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.

Driven by the videogame industry, consumer computer graphics hardware has greatly advanced in recent years, outperforming the computational capacity of central processing units (CPUs). 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 (Luebke and Humphreys, 2007). In contrast to consumer-grade audio cards, GPUs have moved away from the traditional fixed-function pipeline toward a flexible general-purpose computational engine that can currently implement

many parallel algorithms directly using graphics hardware. Current GPUs include fully programmable processing units that support vectorized floating point operations resulting in tremendous computational savings (Owens *et al*., 2007). 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" (Owens *et al*., 2007). GPUs are also becoming faster and more powerful very quickly, far exceeding Moore's Law applied to traditional microprocessors (Ekman *et al*., 1994). 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 (Geer, 2005).

Given the importance of the convolution operation in the generation of spatial audio but its demanding computational requirements, here we present a GPU-based convolution method that is capable of filtering a one-dimensional signal in real-time allowing for the generation of plausible spatial audio for dynamic, interactive applications such as videogames and virtual environments. A comparison with conventional, software-based convolution demonstrates the effectiveness of the developed method.

Paper Organization

The remainder of this paper is organized as follows. The *Background* section provides background information regarding graphic processing units (GPUs) and their use in spatial audio-based applications. The *Implementation* section provides implementation details of the GPU-based convolution method. Performance measures are provided in the *Results* section, where a comparison with conventional, software-based convolution is made and a discussion of the results is also provided. Concluding remarks are presented in the *Conclusions* section.

Background

In this section, an overview of spatial audio and audio processing in general using the GPU is provided. However, prior to doing so, a brief introduction of GPUs is provided.

GPU 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 (Owens *et al*., 2007):

- **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 the "traditional" fixed-function pipelines with "modern" (programmable) GPUs, both the vertex and fragment stages are user-programmable. Programs written to control the vertex stage are known as *vertex programs* or *vertex shaders* while programs written to control the fragment stage are known as *fragment programs* or *fragment shaders*. Early on, these programs were written in assembly language. However, higher level languages have since been introduced including Microsoft's *High Level Shading Language* (HLSL), the *OpenGL Shading Language* (GLSL) (Rost, 2006) and NVIDIA's *Cg (*Mark *et al*., 2003*)*. Generally, the input to both of these programmable stages is a four-element vector where each element represents a 32-bit floating point 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 that typically represent color. The fragment stage is capable of fetching data from texture memory (*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 (i.e., the vertex stage supports both memory gather and memory scatter) (Owens *et al*., 2007). In addition to vertex and fragment shaders, *Shader Model 4.0* currently supported by *Direct3D 10* and *OpenGL 3.0* defines a new type of shader, the *geometry shader*. A geometry shader receives input from the vertex shader and can be used to create new geometry. It is also capable of operating on entire primitives (Sherrod, 2008).

GPU-Based Audio and Spatial Audio Generation

GPUs have also been applied to a wide variety of audio-based applications. von Tycowicz and Loviscach (2008) describe the implementation of a flexible virtual drum that is simulated in real-time and with low latency on the GPU. A MIDI controller with 16 pressure points is used for pressure recognition and a finite difference method is employed to synthesize sound based on location and pressure information. Matsuyama *et al.* (2007), describe a method for the automatic generation of real-time sound for graphics-based animation of sparks to simulate thunder and lighting effects. The implementation also makes use of GPU-based numerical methods introduced by Kruger and Westermann (2003). Using the Cg shading language, Whalen (2005) implements seven common audio functions: *chorus, compress, delay, high-pass filter, low-pass filter, noise-gate* and *normalization*. A performance comparison was made between the GPU and corresponding CPU implementation using a Pentium IV (3.0 GHz CPU) and an NVIDIA GeForce FX 5200 video card. The GPU showed better performance for several of the functions (compress and chorus with speedups of up to a factor of four). However, the CPU implementation was better for other functions (high-pass and low-pass filters). It was suggested that the GPU performance was poorer for some algorithms given the implementation of these algorithms was not suitable for GPU implementation given that they required (computationally expensive) texture access. With more modern video cards, texture access has been improved and this will undoubtedly lead to greater improvements in these methods.

Röber *et al.* (2007) present 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 (Röber, *et al*., 2006). The system was tested on a PC with an AMD64 4000+ dual-core CPU and an NVIDIA GeForce 7900 GT video card and showed speed-ups of factors of from 4.5 to 69 when compared to a software-based implementation. However, the CPU implementation was not optimized.

Audio-based ray tracing using the GPU was implemented by Jedrzejewski (2004) to compute the propagation of acoustical reflections in highly occluded environments and to allow for the sound source and the listener to move throughout the simulation without the need for a long pre-computation phase. Jedrzejewski takes advantage of the fact that in acoustics, as opposed to graphics, objects other than walls do not contribute significantly to the sound wave modifications and therefore can be ignored during the computation. Because of this, only polygons that represent walls are taken into account. Röber *et al*. (2007) 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 ray tracing systems suitably modified to handle sound wave propagation. The system accepts a 3D polygonal mesh of up to 15,000 polygons and pre-processes it into an accessible structure. A frame-rate of up to 25 fps was achieved using a detailed model of a living room containing 1,500 polygons (using an NVIDIA GeForce 8800 GTX video card).

Tsingos and Gascuel (1997) 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 were not considered), absorption, and diffraction caused by partial occluders. Specular reflections are handled using an image source approach (Allen and Berkley, 1979) while diffraction is approximated by computing the fraction of sound that is blocked by obstacles on 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" i.e., not in the background, correspond to occluders). Although their approach is not completely real-time, it is "capable of achieving interactive computation rates for fully dynamic complex environments" (Tsingos and Gascuel, 1997). Tsingos and Gascuel later introduced another occlusion and diffraction method based on the Fresnel-Kirchoff optics-based approximation to diffraction (Tsingos *et al*., 2001; Tsingos and Gascuel, 1998) The Fresnel-Kirchoff approximation is based on Huygens' principle (Hecht, 2002). Given the use of graphics hardware, their method is fast and is well suited to the interactive auralization of diffracted energy maps (Tsingos and Gascuel, 1998). Comparisons for several configurations with obstacles of infinite extent between their method and between boundary element methods (BEMs), gives "satisfactory quantitative results" (Tsingos and Gascuel, 1998). Finally, a complete overview of GPU-based spatial sound and audio processing is beyond the scope of this paper but a thorough review is provided by Hamidi and Kapralos (2009).

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). The input (un-processed) signal can be of any type (i.e., floating point, integer including the "short integer" 16-bit resolution common with WAV files used in many videogames). 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 coefficients can be of any type (e.g., float, integer, etc.). For this work, an NVIDIA GTX-280 video card is being used. Although the implementation is applicable to all video cards, the GTX supports double precision floating point numbers allowing data to be stored with full accuracy thus avoiding the introduction of artifacts in the final (filtered) result (see *Results and Discussion* section).

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. The source (shader) code is provided below. Although the code is made freely available, the authors ask that if used, proper acknowledgment be given. With the GTX-280, the 16-bit (integer) input sound signal is stored in single channel 16-bit intensity maps (images) while the HRTF filter is sent to the GPU as an array of floats (i.e., it is not stored as an image). Although this is not completely necessary and the data can be divided into two 8-bit channels, it does lead to reduction in both computational requirements and round-off errors. A texture is then created from the data in OpenGL. This however is accomplished using the CPU and not the GPU. 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.

Vertex Shader

```
varying vec2 texCoord;

void main(void){
      gl_Position = ftransform();
      texCoord = gl_MultiTexCoord0.xy;
}
```

Fragment Shader

```
uniform vec2 imageSize;
varying vec2 texCoord;
uniform sampler2D image;
const int MAX = 200;
uniform float hrtf[MAX]; //float array sent from CPU

void main (){
      float x = 1.0 / imageSize.x;
      float y = 1.0 / imageSize.y;
      int   length = MAX;
      vec2  currentPos; //position being sampled
      float oldY;
      float total = 0.0; //running total
      vec3   base = vec3(0.0, 0.0, 0.0);
      float temp = 0.0; //used in calculations

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

      int i;
      for(i=0; i<length; i++){
            currentPos.y = oldY + floor(currentPos.x)*x;
            temp = texture2D(image, currentPos).r;
            temp = (temp * 64.0-32.0)* hrtf[i];
            total += temp;
            currentPos.x += x;
      }

      total += 128.0;
      int intTotal = int(total);
      base.r = float(intTotal)/256.0;
      base.g = total-float(intTotal);
      if(total > 128.0){
            base.b = 1.0;
      }
      else{
            base.b = 0.0;
      }

      gl_FragColor = vec4(base,1.0);
}
```

Conversion Code

The code below runs on the CPU and is executed after the vertex and fragment shaders are executed. This code reads the pixel data from the frame buffer and converts it to a 1D array of unsigned integers. The blue channel is used to correct the output in the red channel. More specifically, the blue channel is used to prevent a strange bug in the output from the card. If a

channel outputs a value greater than 128 out of 255, the channel's data is off by one. This can be a large problem when it is the red channel because it is multiplied by 256 meaning that the result would be off by 256. The blue channel acts as a flag to indicate if this has occurred in the red channel so that output can be corrected (this may be video card specific).

```
glReadBuffer(GL_BACK);

glReadPixels(520, 600-wave.length/256-2, tex3.width,wave.length/256+1,
GL_RGB,  GL_UNSIGNED_BYTE, data);

for(GLuint j=0; j<256*(wave.length/256+1); j++){
      Output[j] = data[j*3]*256 + data[j*3+1];
      if(data[j*3+2] >= 127)Output[j]+=256;
}
```

Results and Discussion

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 a filter for each method (the same input signal and HRTF was used for both methods). The filter considered for this test was an actual HRTF filter taken from the CIPIC HRTF dataset, measured from a KEMAR manikin using the "large ear" with the sound source positioned on the horizontal axis directly in front of the KEMAR (Algazi *et al*., 2001). The filter coefficients were floating point numbers (i.e., "float") and of size 200 (although the filter coefficients considered here were floating point values, the proposed method can handle filter coefficients of any type). The input signal was a one-dimensional sine-wave signal (each sample had a resolution of 16 bits and of type "short int"). 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 (2.66 GHz) Processor with 2 GB of RAM and an NVIDIA GTX-280. The GTX supports double precision floating point numbers thus allowing data to be stored with full accuracy.

The results of this test are summarized in Table 1, and Figure 1 where a comparison of the average computational time requirements (x-axis) vs. the size of the input signal of conventional CPU-based (software) convolution and GPU-based (hardware) convolution using both the NVIDIA GTX 8800 and GTX 280 video cards is illustrated. Each point on the graph (both GPU and CPU-based implementations) represents computational time requirements averaged over 1,000 iterations. The GPU-based computational time requirements includes processing on the CPU which was performed to convert the data into the format required by the GPU.

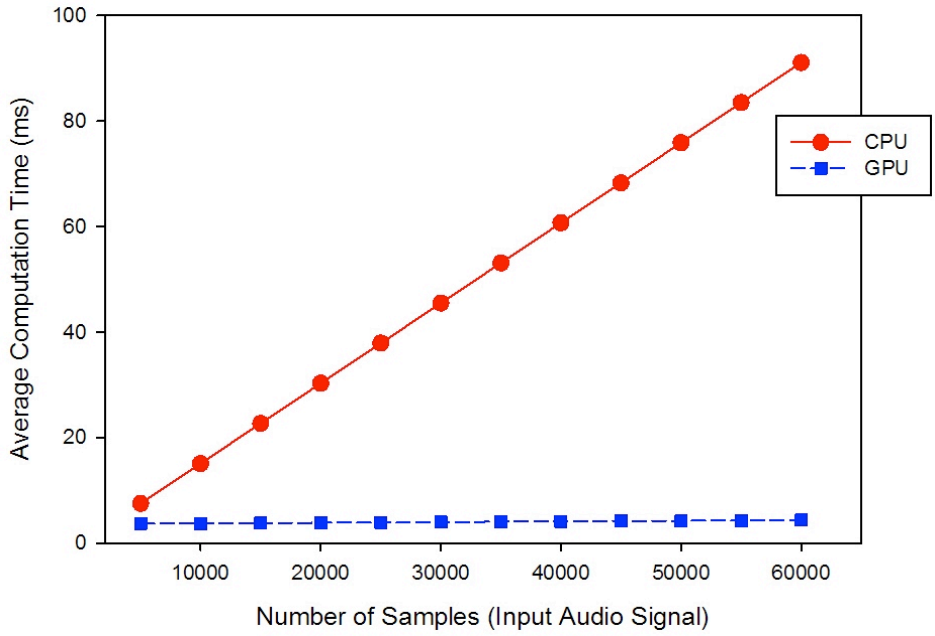

Figure 1: Comparison of input sample size vs. average running time for GPU- and CPU-based convolution. Filter size was constant at 200 samples.

Number of Samples	Time (ms) CPU	Time (ms) GPU
5,000	3.72	0.45
10,000	7.50	0.50
15,000	11.27	0.58
20,000	15.06	0.61
25,000	18.84	0.69
30,000	22.63	0.70
35,000	26.39	0.75
40,000	30.19	0.80
45,000	33.95	0.89
50,000	37.70	0.92
55,000	41.50	0.94
60,000	45.27	0.99

Table 1: Average computational time requirements. The first column represents input signal size (number of samples), the second column represents the average computational time requirements of the conventional CPU-based convolution method, and the third column represents the average computational time requirements GPU-based convolution method.

Discussion

As shown in Figure 1, GPU-based convolution is clearly superior to CPU-based convolution with respect to average computation time. In particular, the average computational running time for the GPU-based method ranged from 0.45 ms (input sample size of 5,000) to approximately 1

ms (input sample size of 60,000) to compute the convolution of an input signal and a filter with 200 coefficients. In contrast, the CPU computational time requirements ranged from 3.72 ms (input sample size of 5,000) to 45.27 ms (input sample size of 60,000). An average computational time of just under 1 ms for the convolution of an input signal with 60,000 samples and an (HRTF) filter with 200 coefficients corresponds to approximately 1,000 fps; clearly applicable for real-time operation. In contrast, the same convolution using the CPU-based method leads to a frame-rate of 22 fps. Furthermore, the NVIDIA GTX-280 video card is over two years old and although at the time of its introduction it was "the fastest single-GPU solution out there", better, more powerful video cards are available that will further reduce the running time requirements. The results presented here are also an improvement from the prior results of Cowan and Kapralos (2009), where a running time of approximately 2 ms was observed for the convolution of an input signal with 60,000 samples and an (HRTF) filter with 200 coefficients. The implementation of that work was optimized leading to the results obtained here.

Given that the convolution operation involves floating-point number calculations, it is important that the video card support double-precision arithmetic to avoid any floating point-related errors which will manifest themselves in the final (filtered) result. As previously described, the NVIDA GTX-280 video card supports double precision computations and therefore floating point errors were not an issue here. However, previous work by Cowan and Kapralos (2008) previously investigated GPU-based convolution using the NVIDIA GeForce 8800 which does not support double-precision arithmetic. The method introduced noise/artifacts to the lower-order bytes of the resulting GPU-based convolution output. This noise resulted from the limitations specifically with the NVIDIA GeForce 8800. More specifically, the GeForce 8800 was returning values with an 8-bit accuracy thus not allowing "images" to have 16-bits per channel. As a result, the 16-bit input sound signal was divided into two 8-bit values (via the red and green channels of the image), combined in the shader and stored as floats. Furthermore, the 8-bits per channel implied that the input had to be divided between two channels. Although this required slightly more computation, it did not interfere with accuracy.

Conclusions

Spatial auditory cues can add a better sense of presence or immersion, compensate for poor visual cues (graphics), and at the very least, add a "pleasing quality" to the simulation. As a result, incorporating spatial audio cues in videogames and virtual environments seems obvious. However, the generation of plausible spatial audio hinges on the convolution operation which itself is computationally expensive thus typically not lending itself to dynamic, real-time applications. To overcome the limitations associated with software-based convolution, here we presented a hardware-based convolution method that takes advantage of the tremendous computational ability of the affordable and commonly available graphics processing units (GPUs). The method was implemented 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 coefficients and a one-dimensional signal of up to 60,000 samples, in real-time (approximately 1 ms). Given that the generation of spatial 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.

Acknowledgements

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 thank the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis, for making their HRTF dataset freely available.

References

Algazi, V. R., Duda, R. O., Thompson, D. M., and Avendano, C. (2001). The CIPIC HRTF database. In *Proceedings of the 2001 IEEE Workshop on Applications of Signal Processing to Acoustics*, New Paltz, NY. USA, October 21-24, 2001.

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

Carlile, S. (1996). *Virtual Auditory Space: Generation and Application*. Austin, TX. USA: R. G. Landes Company.

Cohen, M., and Wenzel, E. (1995). The design of multidimensional sound interfaces. In *Virtual Environments and Advanced Interface Design*, Barfield, W., and Furness, T. Eds. New York, NY. USA: Oxford University Press Inc., pp. 291–346.

Cowan, B., and Kapralos, B. (2008). Spatial sound for video games and virtual environments utilizing real-time GPU-based convolution. In *Proceedings of the ACM FuturePlay 2008 International Conference on the Future of Game Design and Technology*. Toronto, Ontario, Canada, November 3-5, 2008.

Cowan, B., and Kapralos, B. (2008). Real-time GPU-based convolution: A follow-up. In *Proceedings of the ACM FuturePlay 2009 International Conference on the Future of Game Design and Technology*. Vancouver, British Columbia, Canada, May 12-13, 2009.

Durlach, N. I., and Mavor, A. S. (1995). *Virtual Reality: Scientific and Technological Challenges*. Washington, DC. USA: National Academies Press.

Ekman, M., Warg, F., and Nilsson, J. (1994). An in-depth look at computer performance growth, *Computer Architecture News* 33(1):144–147.

Fialka, O., and Cadik, M. (2006). FFT and convolution performance in image filtering on GPU. In *Proceedings of the Conference on Information Visualization*, Washington, DC. USA, June 15-17, 2006, pp. 609-614.

Gardner, W. G. (1995). Efficient convolution without input-output delay. *Journal of the Audio Engineering Society* 43(3):127-136.

Gallos, E., and Tsingos, N. (2003). Efficient 3D audio processing with the GPU. In *Proceedings of the ACM International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 2003)*, San Diego, CA. USA, July 27–31 2003, pp. 896–907.

Geer, D. (2005). Taking the graphics processor beyond graphics, *IEEE Computer* 39(9):14–16.

Hamidi, F., and Kapralos, B. (2009). A review of spatial sound for virtual environments and games with graphics processing units. *The Open Virtual Reality Journal*. 1(1):8-17.

Hecht, E. (2002). *Optics*, fourth ed. Pearson Education Inc., San Francisco, CA. USA.

Jedrzejewski, M. (2004). *Computation of room acoustics on programmable video hardware*, Master's Thesis, Polish-Japanese Institute of Information Technology, Poland.

Kapralos, B., Jenkin, M., and Milios, E. (2008). Virtual audio systems. *Presence: Teleoperators and Virtual Environments*. 17(6):527-549.

Kapralos, B., and Mekuz, N. (2007). 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.

Kistler, D. J., and Wightman, F. L. (1992). 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.

Kleiner, M., Dalenbäck, B., and Svensson, P. (1993). Auralization – an overview. *Journal of the Audio Engineering Society*, 41(11):861-875.

Kruger, J., and Westermann, R. (2003). Linear algebra operators for GPU implementation of numerical algorithms. In *Proceedings of the 30th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 2004)*, San Diego, CA. USA, July 27-31 2003, pp. 908–916.

Luebke, D., and Humphreys, G. (2007). How GPUs work, *IEEE Computer* 40(2):96–100.

Mark, W. R., Glanville, P. S., Akeley, K., and Kilgard, M. J. (2003). 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)*, San Diego, CA. USA, July 27–31 2003, pp. 896–907.

Matsuyama, K., Fujimoto, T. (2007). and N. Chiba, Real-time sound generation of spark

discharge. In *Proceedings of the 15th Pacific Graphics Conference*, Maui, Hawaii, October 29 November 2 2007, pp. 423–426.

Röber, N., Kaminski, U., and Masuch, M. (2007). Ray acoustics using computer graphics technology. In *Proceedings of the 10th International Conference on Digital Audio Effects*, Bordeaux, France, September 10-15 2007.

Röber, N., Spindler, M., and Masuch, M. (2006). Waveguide-based room acoustics through graphics hardware. In *Proceedings of the International Computer Music Conference* 2006, New Orleans, LA. USA, November 6-11 2006.

Rost, R. (2006). *OpenGL Shading Language*, second ed., Addison-Wesley Professional, Boston, MA. USA.

Sherrod, A. (2008). *Game graphics programming*, Course Technology, Cengage Learning, Boston, MA USA.

von Tycowicz, C., and Loviscach, J. (2008). A malleable drum. In *Proceedings of the 35th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 2008 Posters)*, Los Angeles, CA. USA, August 11-15 2008, Article No. 74.

Tsingos, N., and Gascuel, J. D. (1997). Soundtracks for computer animation: Sound rendering in dynamic environments with occlusion. In *Proceedings of Graphics Interface '97* Kelowna, BC. Canada, May 21-23, 1997, pp. 9–16.

Tsingos, N., Funkhouser, T., Ngan, A., and Carlbom, I. (2001). Modeling acoustics in virtual environments using the uniform theory of diffraction. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 2001)*, pp. 545–552.

Tsingos, N., and Gascuel, J. D. (1998). Fast rendering of sound occlusion and diffraction effects for virtual acoustic environments. In *Proceedings of the 104th Convention of the Audio Engineering Society,* Amsterdam, The Netherlands, May 16-19, 1998, pp. 1–14.

Whalen, S. (2005). *Audio and the graphics processing unit*. Author report, University of California, Davis, California, USA.