Virtual Reality (2010) 14:183–196
DOI 10.1007/s10055-010-0166-6

ORIGINAL ARTICLE

GPU-based real-time acoustical occlusion modeling

Brent Cowan · Bill Kapralos

Received: 10 January 2009 / Accepted: 5 July 2010 / Published online: 10 August 2010

Abstract In typical environments, the direct path between a sound source and a listener is often occluded. However, due to the phenomenon of diffraction, sound still reaches the listener by "bending" around an obstacle that lies directly in the line of straight propagation. Modeling occlusion/diffraction effects is a difficult and computationally intensive task and thus generally ignored in virtual reality and videogame applications. Driven by the gaming industry, consumer computer graphics hardware and the graphics processing unit (GPU) in particular, have greatly advanced in recent years, outperforming the computational capacity of central processing units. Given the affordability, widespread use, and availability of computer graphics hardware, here we describe a computationally efficient GPU-based method that approximates acoustical occlusion/diffraction effects in real time. Although the method has been developed primarily for videogames where occlusion/diffraction is typically overlooked, it is relevant for dynamic and interactive virtual environments as well.

CR Subject Classification H.5 INFORMATION INTERFACES AND PRESENTATION, H.5.1 Multimedia Information Systems -Artificial, augmented, and virtual realities. I.3 COMPUTER GRAPHICS, I.3.1 Hardware Architecture -Graphics processors.

B. Cowan (✉) · B. Kapralos
Faculty of Business and Information Technology,
Health Education Technology Research Unit (HETRU),
University of Ontario Institute of Technology,
2000 Simcoe St. North, Oshawa, ON L1H 7K4, Canada
e-mail: brent.cowan@uoit.ca

B. Kapralos
e-mail: bill.kapralos@uoit.ca
URL: http://faculty.uoit.ca/kapralos

Keywords Graphics processing unit · Acoustical occlusion · Acoustical diffraction · Spatial sound · Real-time

1 Introduction

In our natural environment, the direct path between a sound source and a listener is often occluded due to the presence of an occluding object/obstacle in the direct path between the sound source and the listener. However, due to the phenomenon of diffraction, sound is still able to reach the listener by "bending" around the occluder/obstacle. Formally, diffraction can be defined as the "bending mode" of sound propagation whereby sound waves go ("bend") around an obstacle that lies directly in the line of straight propagation, allowing us to hear sounds around corners and around barriers (Cremer and Müller 1978). Diffraction is dependent on both wavelength and obstacle/surface size, increasing as the ratio between wavelength and obstacle size is increased. In other words, diffraction will typically be greater for lower-frequency sounds and when the obstacles are small. The frequency spectrum of audible sound ranges from approximately 20–20 kHz, corresponding to wavelengths ranging from 17 m down to 0.017 m (with a velocity of $v_c = 343\ \mathrm{ms}^{-1}$ for sound in air and a frequency of f Hz, wavelength $\lambda = v_c/f$ (Cremer and Müller 1978)). Since, the dimensions of many of the objects/surfaces encountered in our daily lives are of the same order of magnitude as the wavelength of audible sounds, diffraction is an elementary means of sound propagation especially when there is no direct path between the sound source and the listener, such as in buildings (Tsingos et al. 2001). Given the importance of diffraction to sound propagation and ultimately to human hearing, acoustical occlusion/diffraction effects must be accounted for in videogames and virtual environments. In fact, ignoring occlusion/diffraction effects leads to non-realistic auditory simulations and ultimately results in a decrease in “presence” and immersion (Torres et al. 2001; Tsingos et al. 2001). That being said, modeling occlusion/diffraction effects is a difficult and computationally intensive task and is therefore, typically ignored.

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 3D pipeline toward a flexible general-purpose computational engine that can currently implement many parallel algorithms directly resulting in tremendous computational savings. 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). 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 8 months, far exceeding Moore’s Law applied to traditional microprocessors (Ekman et al. 1994; Geer 2005). Current GPUs are fully programmable and support vectorized floating point operations (Owens et al. 2007). Furthermore, a number of high level languages have been introduced to allow for the control of vertex and pixel pipelines (Buck et al. 2004). 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) (Owens et al. 2007). A thorough review including a detailed summary of GPGPU-based applications is beyond the scope of this paper but is given by Owens et al. (2007) and will therefore not be provided here.

Given the potential computational efficiencies that may be obtained with GPUs, we have begun an initiative to employ the GPU to generate spatial (3D) audio with the ultimate goal being the generation of accurate spatial audio in real time for inclusion in videogames and virtual environments (an overview of auditory perception and spatial audio is provided in Kapralos et al. (2008b)). In this paper, we describe the development of a novel, computationally efficient GPU-based method that approximates occlusion/diffraction effects in real time. The method has been developed primarily for videogame applications (where occlusion/diffraction is typically completely overlooked) that are dynamic and interactive, and require real-time operation. Rather than focusing on faithfully modeling physical acoustical diffraction models, emphasis is placed on computational efficiency; the method approximates acoustical occlusion/diffraction in real time regardless of scene complexity yet it conforms to theoretical diffraction models that dictate diffraction decreases with increasing frequency. In other words, a trade-off between accuracy and efficiency is made. Although the method has been developed primarily for videogames, it is relevant for dynamic and interactive virtual environments where real-time operation is required and an accuracy vs. efficiency trade-off can typically be made.

The remainder of this paper is organized as follows. Section 2 provides background information. The section begins with a brief discussion of GPUs followed by an overview of existing acoustical diffraction modeling techniques (both GPU- and non-GPU-based methods are described). Details regarding the GPU-based occlusion modeling method presented here are provided in Sect. 3. In Sect. 4, the results of a number of simulations that illustrate that the method conforms to real-world sound propagation properties are provided. Finally, a summary and plans for future work are presented in Sect. 5.

2 Background

A number of research efforts have examined the generation of spatial sound using the GPU. An overview of these methods is beyond the scope of this report but is available in (Hamidi and Kapralos 2009). In this section, several research efforts that have focused on acoustical occlusion/diffraction modeling both with and without the use of the GPU are described. However, prior to doing so, 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 (Owens et al. 2007):

- Vertex Stage: (1) Transformation of each (object space) vertex into screen space, (2) formation of triangles from the vertices, and (3) per-vertex lighting calculations.
- Rasterization Stage: (1) Determination of the screen position covered by each of the triangles formed in the previous stage and (2) 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).

2.2 Acoustical occlusions and diffraction modeling with and without the GPU

The beam tracing acoustical modeling approach of Funkhouser et al. accounts for diffraction using a method based on the uniform theory of diffraction (UTD) (Funkhouser et al. 2004; Keller 1962; Tsingos et al. 2001). Validation of their approach by Tsingos et al. involved a comparison between actual measured impulse responses in a simple enclosure (the "Bell Labs Box") and the impulse responses obtained by simulating the enclosure (Tsingos et al. 2002). Sonel mapping is a probabilistic-based acoustical modeling method that is based on photon mapping, a popular image synthesis method (Jensen 2001). Sonel mapping accounts for occlusion/diffraction effects using a modified version of the Huygens-Fresnel principle (Kapralos et al. 2008a). Quantitative results indicate that sonel mapping conforms to theoretical acoustical diffraction models and being probabilistic based, it allows for an accuracy versus efficiency trade-off to be made. However, despite the computational improvements, sonel mapping is not applicable for real-time applications except when considering very simple environments. Tsingos and Gascuel developed an occlusion and diffraction method that utilizes computer graphics hardware to perform fast sound visibility calculations that can account for specular reflections, absorption, and diffraction caused by partial occluders (Tsingos and Gascuel 1997). 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 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" i.e., not in the background, correspond to occluders). Their approach handles a discrete set of frequency bands ranging from 31 Hz to 8 kHz 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 takes advantage of graphics hardware to perform this in an efficient manner. Their approach is not completely real time, but 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-Kirchhoff optics-based approximation to diffraction (Tsingos and Gascuel 1998). The Fresnel-Kirchhoff approximation is based on Huygens' principle (Hecht 2002). The total unoccluded sound pressure level at some point p in space is determined by calculating the sound pressure of a small differential area dS and integrating over the closed surface enclosing p (see Tsingos and Gascuel for further details regarding this calculation in addition to an algorithm outlining the method (Tsingos and Gascuel 1998)). After determining the total unoccluded sound pressure arriving at point *p* from a sound source, diffraction and occlusion effects are accounted for by computing an occlusion depth-map of the environment between the sound source and the receiver (listener) using computer graphics hardware to permit real-time operation. Once the depth-map has been computed, the depth of any occluders between the sound source and the receiver can be obtained from the Z-buffer (Foley et al. 1994) whereby "lit" pixels correspond to occluded areas. The diffraction integral described by the Fresnel-Kirchhoff approximation is then approximated as a discrete sum of differential terms for every occluded pixel in the Z-buffer. Comparisons for several configurations with obstacles of infinite extent between their method and between boundary element methods (BEMs) give "satisfactory quantitative results" (Tsingos and Gascuel 1998). Tsingos et al. (2007) describe a high quality, GPU-based acoustical first-order sound scattering modeling method that is based on a surface integral formulation and Kirchhoff's approximation. Their method is capable of modeling both diffraction and reflection in an arbitrarily complex environment. Experiments indicate their method fares well with boundary element methods (BEMs) although greater work remains to allow for higher-order sound scattering and to overcome the fact that the method is prone to aliasing.

Various other research efforts have examined non-geometric acoustics-based diffraction modeling. Torres et al. (2001) describe a time-domain model based on the Biot-Tolstoy-Medwin technique (Biot and Tolstoy 1957), which computes edge diffraction components and combinations of specular and diffracted components. Lokki et al. (2002), Calamia et al. (2005), and Svensson et al. (1999) have also investigated diffraction modeling based on the Biot-Tolstoy-Medwin technique. The method of Calamia et al. (2005) provides an integrated approach to acoustical modeling whereby intermediate values typically used in diffraction calculations using the Biot-Tolstoy-Medwin technique are exploited to find specular reflections as well. In addition to these techniques, a number of wave-based acoustical modeling methods, whose objective is to solve the wave equation in order to recreate the room impulse response that models a particular sound field, inherently account for occlusion/diffraction effects (see Murphy and Beeson 2003 for example). Such techniques are currently not generally applicable to real-time, interactive applications due to complexity issues and are therefore not considered further here.

In contrast to previous approaches, the proposed method can model multiple occlusions (i.e., several "layers" of occluding obstacles) between the sound source and the listener. Essentially, this can be viewed as an extension to the approach of Tsingos and Gascuel (1998) that employs a more physically based approach but is limited to surfaces directly visible from the source. Furthermore, rather than focusing on the faithful physical modeling of acoustical occlusion/diffraction effects, the proposed method is intended to approximate these effects but at real-time (interactive) rates regardless of scene complexity. That being said, as will be described in Sect. 4, although only an approximation, diffraction behavior is evident.

Finally, with respect to video games, a number of APIs and specifications are available that do account for occlusion effects. The Creative Labs Environmental Audio Extensions (EAX) is a set of extensions for Microsoft's DirectSound3D API that allow for adjustment of sound source and listener parameters and occlusion effects. Occlusion effects are divided into three categories (Menshikov 2003): (1) *occlusions*, where the sound bends around an insurmountable obstacle that divides the sound source and the listener, (2) *obstructions*, where the sound source and the listener are in the same room, the direct path is blocked but reflected sound still reaches the listener, and (3) *exclusions*, where the sound source and listener are in different rooms but there is an opening between the rooms and the direct sound and only partial reflections reach the listener. Each of the three effects is accounted for with a low-pass filtering operation. For occlusions and reflections, the resulting sound is essentially muffled using a low-pass filtering operation. With obstructions the direct path sound is muffled while the reflections are not altered. With exclusions, the direct sound reaches the listener and the reflections are distorted. FMod is an audio API for the creation and playback of interactive (positional) audio that also supports occlusion modeling. In the FMod specification, occlusion is described as the changes in volume, frequency content, and reverberation that occur when sound passes through an object. Occlusion is accounted for by attenuating the volume of the direct and reverberant sound (when the direct path between the sound source and the listener is occluded), and attenuation of the high-frequency components of both the direct and reverberant sound using a low-pass filtering operation (Menshikov 2003). OpenAL is another commonly used audio API and when it implements the Environment Effects Extension (EFX) (that provides the same functionality as EAX but tailored to OpenAL), and it also supports occlusion mapping using simple low-pass filtering operations (Menshikov 2003). Section 4.4 provides details regarding an interactive demo available online that implements occlusion modeling of a particular environment using FMod and the proposed method. Informal listening tests indicate that the proposed method is capable of providing more convincing results.

3 Methods

In this section, implementation details of the GPU-based occlusion modeling method (i.e., the shader) are provided. The method uses the actual scene geometry, including moving objects, to approximate occlusion/diffraction effects. Rather that attempt to build a complete sound engine, here we focus on approximating occlusion effects and passing the results to an existing sound engine in order to increase realism. The implementation is based on the *OpenGL shading language* (GLSL) (Rost 2006) (see Appendix A.1 and A.2 for the GLSL shader source code). The input to the resulting shader is the scene/model containing the sound source and listener/receiver. The scene itself can be arbitrarily complex (i.e., any number of objects) although for illustrative purposes, the scene for any examples presented here have been purposely kept simple.

For computational efficiencies, the distribution of sound frequency in a given sound source is typically approximated by a fixed number (N_{freq}) of frequency bands (Mehta et al. 1999). Here, only two frequency bands are considered; low and high frequencies. Low frequencies are defined to be less than 4 kHz, and high frequencies are greater than or equal to 4 kHz. However, as will be described later, the proposed occlusion modeling method operates in real time; average running time of 0.39 ms irrespective of the scene complexity. The method can, therefore, be easily extended to account for a greater number of frequency bands and considering only two frequency bands is purposely done to simplify the explanation and result illustrations. The method is completely dynamic and operates irrespective of the sound source, the listener, and object position and orientation; the sound source, the listener, and objects can freely move about within the environment. The proposed method is a two-stage technique; *acoustical ray casting* followed by *receiver scaling*. Both stages are detailed in the following sections.

3.1 Acoustical ray casting stage

The purpose of the acoustical ray casting stage is to produce a *listener occlusion map* which, for every position in the scene (pixel), provides an occlusion weighting ($w_{\text{occlusion}}$) within the range of [0…1]. 0 (“black”) indicates the sound source is completely blocked (non-visible) and no sound reaches the listener, while 1 (“white”) indicates the sound source is completely visible to the listener and direct sound is able to reach the listener. Shades of gray in between represent partial occlusion whereby some of the sound is able to reach the listener. The scene is rendered (using the GPU) from the sound source’s perspective. To construct the occlusion map, we initially begin with an empty (“white”) canvas. A total number (N_{vis}) of visibility rays are emitted from the sound source to the receiver in order to determine the visibility between the sound source and the receiver (i.e., the visibility between the sound source and the receiver is determined by calculating the number of rays that that are not occluded and can therefore reach the listener) using the GPU. Basically, a three-dimensional area between the sound source and the listener is sampled using ray-casting. This area represents a view frustum with the apex located at the sound source and the far viewing plane centered on the listener’s position. The number of visibility rays emitted depends on the resolution of the occlusion map. More specifically, one ray is emitted for each pixel in the occlusion map. The size of the occlusion map is currently set to 128 × 64 ($N_{\text{vis}} = 8{,}192$ rays) and although the size can be adjusted, increasing the size of the occlusion map does not improve the result noticeably but it does increase the computational requirements. That being said, a very low-resolution occlusion map (e.g., 32 × 16) does not closely represent the shape of the occluding objects. The occlusion map is recomputed every time a sound source, listener, or object in the environment moves.

Since the scene is being rendered from the sound source’s position, the sound source will be “in front of” any surface being rendered. The dot product o_s of the surface’s normal s_{n} and a “test vector” v_t (a normalized vector starting at the listener and facing the point on the surface currently being rendered) is calculated to determine the amount of occlusion

$$o_s = v_t \cdot s_n \tag{1}$$

when $o_s \leq 0$, the listener is in front of the surface being rendered and the surface does not occlude any sound, and therefore, o_s is assigned a value of 0 (i.e., $o_s = 0$). If the surface’s normal s_n is facing away from the sound source, then the surface is not rendered due to back face culling. When $o_s > 0$, rather than assuming the surface occludes 100% of the sound, the occlusion weighting ($w_{\text{occlusion}}$) is assigned the value returned by the dot-product operation ([0 … 1]). The scene is rendered using a perspective projection relative to the sound source. To calculate the occlusion/diffraction between one sound source and one listener, the camera is placed at the location of the sound source and pointed toward the listener. The distance of the far clipping plain is set to match the listener’s distance from the sound source so that the listener is located in the center of the flat bottom of the view frustum (the frustum has a flat bottom but the shader discards fragments beyond a certain distance thus rounding the bottom). Thus, the listener, which is not rendered, is in the exact center of the rendered scene. An occlusion value is calculated for each pixel in the render by the shader based on the occluding geometry.

Rendering the scene using a perspective projection causes the sudden and complete occlusion of the sound when the "camera" (representing the sound source) is close to an occluding object. Rendering the scene orthographically will correct this problem but will subsequently introduce new problems. More specifically, rendering a large enough view orthographically will cause the near clipping plane to intersect solid objects but, since the back facing surfaces are not typically drawn, objects that should occlude would be depicted as empty space. To account for this "camera blocking" problem, when the distance between the sound source and the surface/object (d_s) is less than a pre-defined amount δ, the sound is further scaled as follows

$$o_s = o_s \times \min(d_s/\delta, 1.0). \tag{2}$$

when $d_s < \delta$, the sound level is reduced in a distance-dependent manner such that the closer the object/surface is to the sound source, the lesser the reduction. If $d_s = 0$ (i.e., the object is touching the sound source), the sound is allowed to "pass through" without being reduced. Allowing sound to pass through objects that are close to the camera in this manner is only an approximation and not necessarily based on real-world acoustics. However, it can be computed quickly and it prevents small objects very close to the camera from completely blocking sound, which would be unrealistic and noticeable.

Once o_s is computed, it is subtracted from the current occlusion map pixel value (i.e., subtracted from the scene using OpenGL blending). The order that objects are subtracted does not matter. Beginning with a scene that is completely open (i.e., a "white" occlusion map), layer after layer of occluding objects are added by subtracting the occlusion weighting ($w_{\text{occlusion}}$) of the object, on a pixel-by-pixel basis, from the corresponding pixel value in the occlusion map, simulating the fact that overlapping occluding objects have a cumulative effect on occlusion. A perfect occluder is represented in the occlusion map as solid black while partial occluders are represented partially transparent. When multiple partial occluders are placed one in front of the other, they collectively cause the occlusion map to become darker. The occlusion data are then placed in the green channel, one byte per pixel (the red and blue channels are currently not used). An example illustrating the construction of the occlusion map in a "step-by-step" basis is illustrated in Fig. 1 (for illustration purposes, occlusion is represented in shades of blue).

3.2 Receiver scaling

The output of the acoustical ray casting stage is the gray-scale occlusion map. The occlusion map provides, from the sound source's perspective, a quick approximation to the amount of sound that will reach the listener. The occlusion map is then scaled using a 2D scaling filter. The size of the scaling filter is identical to the size of the occlusion map (i.e., 128 × 64 pixels), and the scaling is performed by overlaying the scaling filter on top of the occlusion map (since the scene is rendered from the sound source to the listener, the listener is always positioned exactly in the center of the render, and likewise is always positioned in the center of the scaling filter) and multiplying each pixel of the occlusion map by its corresponding scaling filter value

$$O_{\text{scaled}}(i,j) = \left(O_{\text{map}}(i,j) \times f(i,j)\right), \tag{3}$$

where $O_{\text{scaled}}(i, j)$ is the value of the scaled pixel at location i, j, $O_{\text{map}}(i, j)$ is the occlusion weighting factor ($w_{\text{occlusion}}$) of the pixel in the occlusion map pixel at location i, j, and $f(i, j)$ is the scaling filter coefficient value at location i, j. After the occlusion map has been filtered, the values of all the pixels in the map are summed and this sum is then scaled by the sum of all filter coefficients, yielding a single floating point value ranging between 0 and 1 and denoted by s_{map}

$$s_{\text{map}} = \frac{\sum_{i,j} O_{\text{map}}(i,j)}{\sum_{i,j} f(i,j)}. \tag{4}$$

As Hillesland and Lastra (2004) describe, although GPUs have floating point representations similar to and at times matching the IEEE standard, GPUs do not adhere to the IEEE standards (IEEE 1987) and do not provide the necessary information to establish a bound on floating point operation errors. Therefore, given that the scaling filters contain "very small" floating point values, to avoid potential floating point errors, the filtering operation is performed using the CPU. Since the scaling filters remain constant, they are calculated once during the "initialization period" and re-used as needed; the filtering operation requires minimal computational resources. Scaling filter coefficient values range from one (center of the scaling filter) to zero (each of the four corners). Each "pixel" within O_{scaled} represents the sound level for a corresponding location in three-dimensional space. The scaling filter is used to approximate the dependency of frequency and obstacle size on diffraction. This filtering operation is specific to one sound source and one receiver pair and must be repeated for each sound source/receiver combination.

Currently, frequency-dependent effects of diffraction are approximated using two scaling filters only; one for high frequencies (≥4 kHz) and another for low frequencies (<4 kHz) (see Appendix A.3 for the source code used to build both the low- and high-frequency scaling filters).

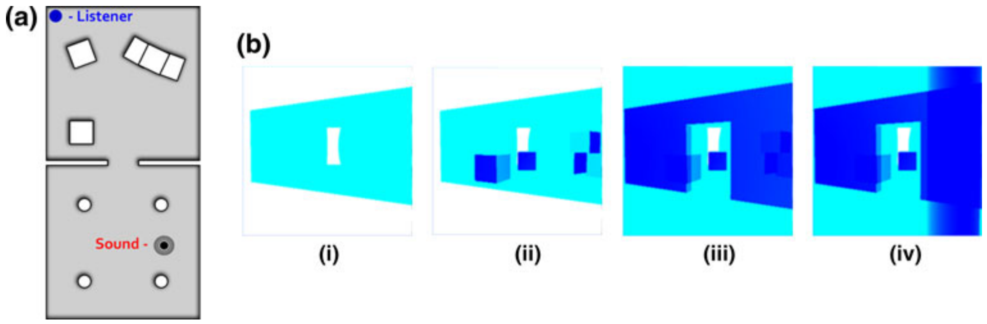

Fig. 1 Sample occlusion map construction. **a** Original scene consisting of a sound source, a listener, and multiple partial occluders. **b** The scene is rendered from the sound source to the listener. Beginning with (*i*), objects are added to the render until the complete scene with all of the objects is rendered in (*iv*). When multiple partial occluders are placed one in front of the other, they collectively cause the occlusion map to darken

A graphical illustration of low- and high-frequency scaling filters is illustrated in Fig. 2a and b, respectively. The low-frequency scaling filter contains a wider lobe ensuring that a wider area of sound reaches the listener (i.e., a greater number of non-zero scaling filter coefficients). In contrast, the high-frequency scaling filter has a narrower lobe thus blocking a greater amount of the overall sound given that a large number of the scaling filter coefficients are zero. Currently, the parameters specifying the lobe width of the scaling filters have been determined through "trial and error" and through informal listening tests.

Once the scaling has been performed, the sound level for a particular location in the three-dimensional space can be scaled to account for distance-dependent attenuation affects by the medium (air). Assuming planar sound waves, the attenuation of sound energy due to absorption by the air follows an exponential law (Kuttrff 2000)

$$E_r = E_o e^{-mr}, \tag{5}$$

where E_o is the original sound energy, E_r is the energy after the sound has traveled a distance r, and m is the air absorption coefficient that varies as a function of the conditions of the air itself (e.g., temperature, frequency, humidity, and atmospheric pressure). Expressions for the evaluation of m are provided by Bass et al. (1990). Alternatively (and as done in this work), rather than explicitly attenuating the sound to account for attenuation effects by the medium, the values from the occlusion map can be passed to the audio engine (e.g., FMod as used in this work, OpenAL, etc.) where further processing can be performed including distance-dependent attenuation.

4 Results: comparisons to real-world acoustical properties

Several simulations were performed with various sound source, receiver, and environmental configurations to demonstrate that the proposed acoustical occlusion modeling method conforms to real-world acoustical occlusion/diffraction models. Although desirable, validation involving human user tests is beyond the scope of this work. Validation that involves comparisons between measurements made in an actual (controlled) room and the results of the same measurements made in a simulation of the room are also not included. Such tests require the use of a controlled room (environment), where various parameters (e.g., surface reflection and absorption coefficients) can be

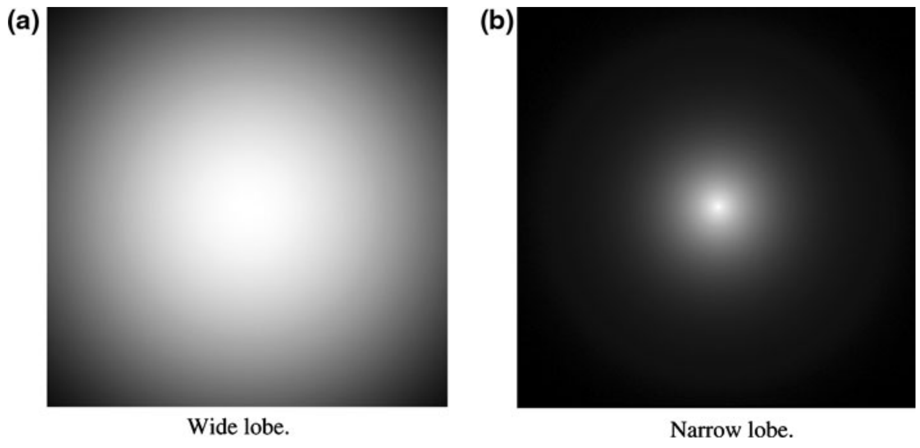

Fig. 2 2D sinusoidal scaling filters used to filter (scale) the occlusion map. **a** Low-frequency (*wide*) lobe filter and **b** high-frequency (*narrow*) lobe filter

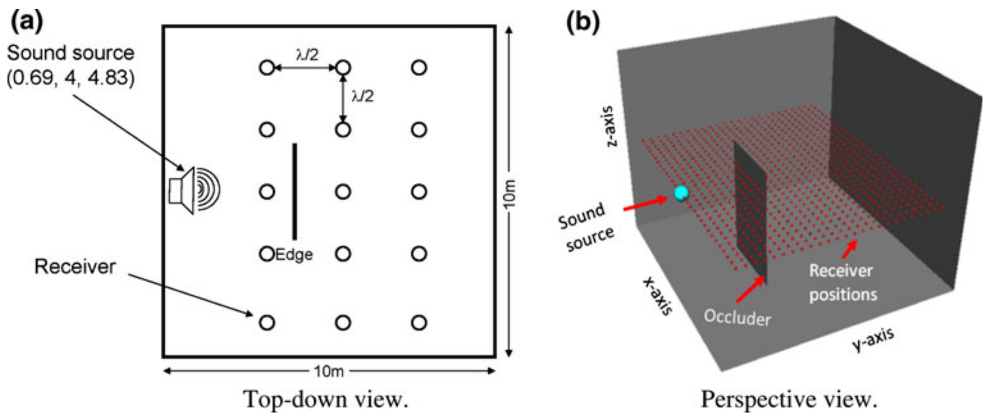

Fig. 3 Room set-up for the individual component graphical demonstrations (not to scale). The height (z-coordinate) of the sound source and the receiver positions was constant at $z = 4$. Receiver positions varied across the x–y plane in equal increments of $\lambda/2$ or 0.685 m (the x coordinate ranged from 1.37 to 8.926 m while the y coordinate ranged from 0.345 to 8.97 m)

carefully controlled in order to allow meaningful comparisons to be made. Given the lack of such available data for a simple room whose parameters can be easily controlled, constructing such an environment is also beyond the scope of this work.

All the simulations described in this section were performed using a Dell XPS 720 workstation with an Intel Quad Core 2.4 GHz processor, 4 Gb RAM, Sound Blaster X-Fi sound card, and an NVIDIA GeForce GTX 280 video card (over-clocked, 1 GB-GDDR3 SDRAM) that supports double precision floating point numbers. The size of the occlusion map was 128 × 64 and in the ray-casting stage, a single ray was cast for each pixel of the occlusion map.

4.1 Graphical Illustrations

In this section, the results of modeling the occlusion effects of an environment (room) consisting of a sound source, receiver, and occluder configuration are presented graphically. As shown in Fig. 3, the dimensions of the room were 10 m × 8 m × 10 m. A low-frequency (500 Hz with $\lambda = 0.685$ m) sound source was used (although only low and high frequencies are considered by the method, a 500 Hz source was used to ensure comparisons could be made with the sonel mapping approach as described later). The sound source was positioned at location (0.685, 4.0, 4.83) and remained stationary while the position of the receiver was varied across the x–y plane (i.e., z-axis remained constant at $z = 4$) in equal increments equal to $\lambda/2$ or 0.685 m (the x coordinate ranged from 1.37 to 8.926 m, while the y coordinate ranged from 0.345 m to 8.97 m). The dimensions of the occluder were 3.0 × 5.0 m and the occluder was positioned such that it formed a plane in the x–z axis. The coordinates of the vertices comprising the occluder were (3.45, 0.0, 3.45), (3.45, 5.0, 3.45), (3.45, 0.0, 6.45) and (3.45, 0, 6.45).

The results of this simulation are shown in Fig. 4a. Sound level (power) is coded in shades of gray where white denotes full sound level (90 dB), black complete silence (0 dB), and shades of gray between represent levels in-between. In this example, for the purposes of illustration, attenuation effects of both the air and distance traveled were completely ignored. As shown in Fig. 4a, although the direct path between the sound source and the listener is completely blocked, sound is still able to reach the listener due to diffraction. As a comparison, using the same room, sound source, and receiver configuration, the occlusion effects were modeled using the sonel mapping method that accounts for occlusion/diffraction effects using a modified version of the Huygens-Fresnel principle (Kapralos et al. 2008a) (see Fig. 4b). A comparison between the results obtained with both approaches illustrates that both methods allow sound to be diffracted so that sound does reach the area behind the occluder although the shadow region is far more pronounced for the sonel mapping method. In contrast, the shadow region for the occlusion modeling method presented here is smoother but extends for a larger region beyond the occluder.

4.2 Graphical comparison to the “base-case”

In this section, the effectiveness of the occlusion mapping method is presented visually by comparing it to the “base-case” ray-casting method, whereby a single “visibility ray” from the sound source to the listener (or from the listener to the sound source) is cast to test for occlusion. If the ray is unobstructed, the sound is played normally. However, if the visibility ray encounters an object, it is assumed that the sound is completely blocked and therefore, the corresponding sound is not played (i.e., silence). In other words, occlusion is treated as a binary state: the sound is either ON or OFF. Although such an approach is simple, it is problematic for a number of reasons including the fact that switching sounds between these two states is noticeable and unrealistic (e.g., consider a listener standing in front of an open doorway; sound emanating from inside the room may be completely unoccluded but if the listener were to take one step to the side, the sound may instantly switch to the occluded state).

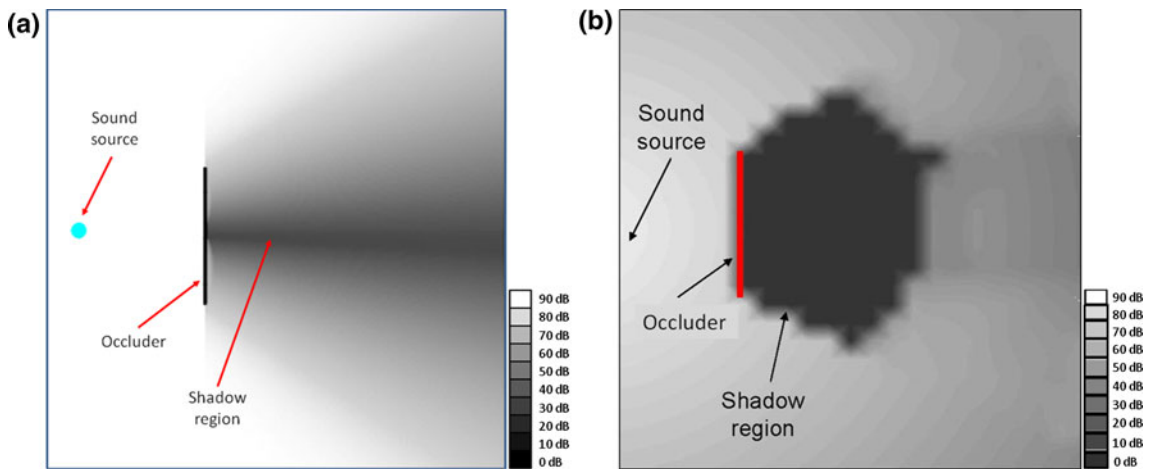

Fig. 4 A comparison of the proposed occlusion modeling method and the sonel mapping method that accounts for occlusion/diffraction effects using a modified version of the Huygens-Fresnel principle (Kapralos et al. 2008a). The room configuration illustrated in Fig. 3 was simulated using both methods. Results using the proposed GPU-based occlusion modeling method are shown in (**a**), and results using the sonel mapping method are shown in (**b**). For both simulations, sound level (power) is coded in *shades of gray* where white denotes full sound level, *black* complete silence, and *shades of gray* between represent levels in-between

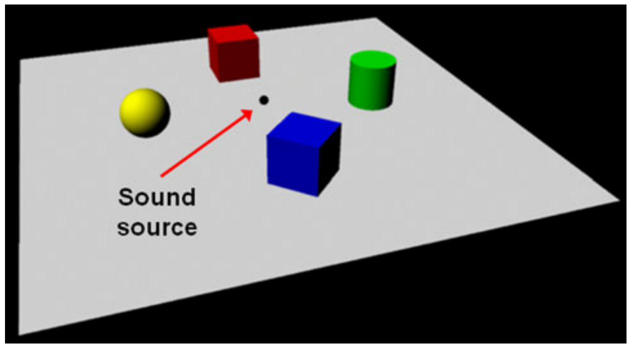

Fig. 5 A simple scene consisting of four objects (two cubes, a sphere, and a cylinder) and a sound source

The scene illustrated in Fig. 5 consisted of four objects (two cubes, a sphere, and a cylinder), and a sound source positioned in the center was rendered using both the base-case approach and the proposed occlusion modeling method. The dimensions of each of the two (red and blue) cubes were 10 m × 10 m × 10 m, the (green) cylinder was 10 m high with a 10 m diameter, while the (yellow) sphere had a diameter of 10 m. The center of each of these objects was placed 25 m from the sound source. All objects rested on a plane measuring 100 m × 100 m and the sound source was positioned 4 m above this plane. The volume was sampled at a constant height of 4 m across the entire plane. A low-frequency sound source was used (250 Hz with $\lambda = 1.37$ m).

For each scenario and method tested, once the occlusion map was constructed, stage two (receiver scaling) was performed for every position in the scene, leading to the graphical output illustrated in Fig. 6a ("base-case approach") and Fig. 6b ("proposed method"). As with the simulation described in the previous section, sound level (power) is coded in shades of gray where white denotes full sound level, black complete silence, and shades of gray represent levels in-between. In this example, the far clipping plane was set equal to the distance of the listener thus objects behind the listener were not considered. Attenuation effects introduced by both the air and distance traveled were completely ignored for purposes of illustration (so that occlusion effects are not confused with attenuation resulting from sound propagating through the air and with distance). As shown in Fig. 6a, with ray-casting of the base-case method, diffraction effects were not accounted for and thus, if the direct line-of-sight between the sound source and receiver is occluded, the receiver will be in complete silence despite the fact that in our natural environment, sound will reach the receiver via diffraction. In contrast, as shown in Fig. 6b, although the sound level was reduced, the sound still reached the shadow regions despite the occlusion of the direct line-of-sight between the sound source and the receiver, better representing real-world sound propagation.

To illustrate the conformance of the method to real-world sound propagation models, this simulation was repeated twice. In the first case, all parameters remained the same except for the dimensions of the four obstacles which were doubled. The updated object dimensions were changed as follows: the dimensions of the two (red and blue) cubes were 20 m × 20 m × 20 m, the (green) cylinder was 20 m high with a 20 m diameter, while the (yellow) sphere had a diameter of 20 m. The center position of each of these objects remained the same at 25 m from the sound source. Given the increase in obstacle size (while sound source frequency remained the same i.e., 250 Hz), less sound should be diffracted. In the second case, all parameters remained the same as the original simulation (the

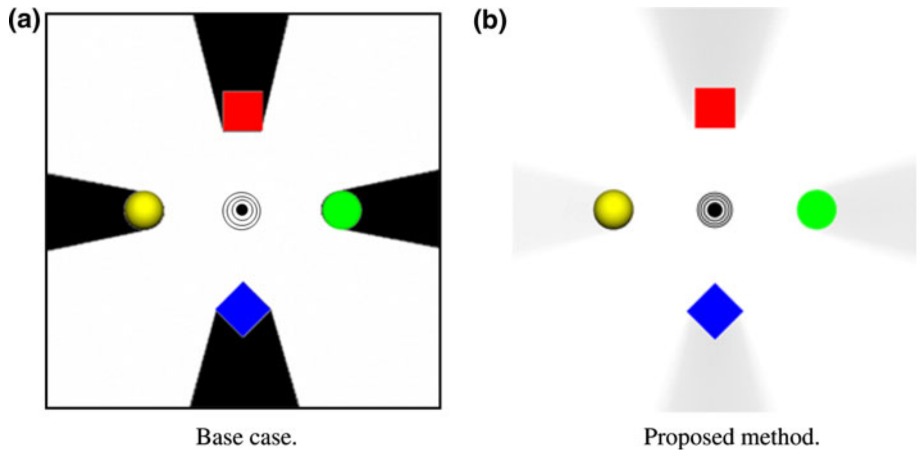

Fig. 6 Acoustical occlusion modeling. **a** Base-case acoustical occlusion approximation. Computing occlusion effects by rendering from the sound source's perspective using ray casting (single ray) while completely ignoring all reflection phenomena. **b** Proposed method

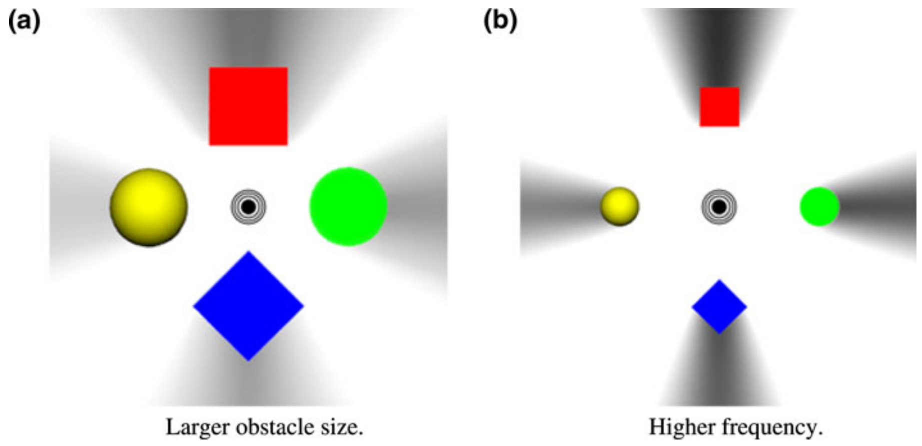

Fig. 7 Acoustical occlusion with the proposed method. **a** Increasing (doubling) the size of the obstacles and **b** increasing (doubling) the frequency

dimensions of each of the two (red and blue) cubes were 10 m × 10 m × 10 m, the (green) cylinder was 10 m high with a 10 m diameter, while the (yellow) sphere had a diameter of 10 m) except that a high-frequency (4 kHz) sound source was used. Given the inverse relationship between diffraction and frequency, increasing the frequency of the sound source should lead to a decrease in diffraction.

The results of this test are illustrated in Fig. 7. As shown in Fig. 7a, when the size of the obstacles was increased, it is clear that less sound was diffracted into the shadow region (the shadow region is darker that the corresponding shadow region illustrated in Fig. 6b where the dimensions of the four objects were halved). Similarly, by increasing the frequency of the sound source, as shown in Fig. 7b diffraction was decreased allowing less sound to reach the shadow region (the shadow region is darker that the corresponding shadow region illustrated in Fig. 6b where a low-frequency (250 Hz) sound was used).

4.3 Running time requirements

As a measure of running time requirements, the scene consisting of the four objects illustrated in Fig. 5 was rendered a total of 20,000 times (rendered 4,000 times at five different listener positions). The average running time (averaged over the 20,000 renders) to render the scene was 0.39 ms. The difference between the running time of each of the 20,000 renderings was insignificant and a plot of rendering vs. running time essentially yields a straight line. Each rendering basically represents the time required to calculate occlusion/diffraction for one sound source and one listener. Running time included the time required for both the acoustical ray-casting and receiver scaling stages. The scene itself was rendered as a "single model". In other words, no scene graph was used. As an aside, to produce the resulting plots of the simulation outlined in the previous section (the four objects and sound source), the rendering has to be repeated for every position/pixel (i.e., every pixel represents a listener).

4.4 Interactive demo

An interactive first-person style demo that demonstrates the operation of the proposed method and provides a comparison between the proposed method and the occlusion modeling capabilities of the FMod audio API is available online via the following website: http://faculty.uoit.ca/kapralos/OcclusionModeling. The demo consists of a simple environment with two rooms joined by an open doorway (see Fig. 1a). Each room contains occluding obstacles such as boxes and pillars. A sound source plays (continuously) a looping snippet of music. Using the standard arrow keys, the "player" is free to move about the environment, sampling the effect that obstacles have on the sound they hear. Press "1" to use the proposed occlusion modeling method (the demo begins in this state), press "2" to use FMod's occlusion modeling method. Moving the mouse rotates the camera (standard FPS controls), and pressing "Esc" exits the demo.

5 Conclusions

In this paper, we have presented a GPU-based occlusion modeling method capable of approximating plausible acoustical occlusion/diffraction effects in real time for use in interactive and dynamic videogames and virtual environments. The method is implemented using the OpenGL shading language thus allowing it to be employed on commonly available programmable GPU video cards. Given the importance of diffraction with respect to sound propagation in our natural environment and the widespread availability of computer graphics cards with onboard programmable GPUs, occlusion/diffraction effects can now be easily and efficiently accounted for in videogames and virtual environments. Despite being an approximation to acoustical occlusion/diffraction, the results of several simulations indicate that diffraction behavior is evident and more specifically, diffraction effects increase as obstacle size decreases and/or frequency decreases.

Hearing is a perceptual phenomenon, and therefore, future work will investigate the effectiveness of the method presented here with human participants in the form of listener tests. The results of such tests will provide greater indications regarding the effectiveness of the method in addition to providing further insight which may lead to corresponding changes and improvements.

Acknowledgments The financial support of the Natural Sciences and Engineering Research Council of Canada (NSERC) in the form of an Undergraduate Summer Research Award and a Post Graduate Scholarship to Brent Cowan and a Discovery Grant to Bill Kapralos is gratefully acknowledged.

Appendix A

In this section, the GLSL vertex and fragment shader source code in addition to the source code to implement the scaling filters is provided (the scaling filters are generated using the CPU once during initialization; the scaling filtering is performed on the CPU).

A.1 Vertex shader

```
/*
Brent Cowan & Bill Kapralos, June 2009

This shader samples many points between the sound source and listener gathering information
much like a person taking a survey. The data collected is rendered as an image to be read back
from the frame buffer. The image is then filtered (scaled) on the CPU in order to estimate the
occlusion, diffraction, and reverberation present in the sampled space.
*/

//The distance between the sound source and listener could be calculated on the CPU more
//efficiently.
//These variables have been provided more for future use.
uniform vec3  soundPos;   //Sound source position xyz
uniform vec3  listenPos;  //Listener position xyz

varying vec3  normal;     //Stores the interpolated surface normal
varying vec3  position;   //position of the object's surface currently being rendered
varying float distance;   //distance between the sound and listener
varying vec3  testVector; //a normalized vector pointing from the listener to the point on
                          //the object currently being rendered

void main ()
{
       gl_Position = ftransform();
       normal = normalize( gl_NormalMatrix * gl_Normal );
       position = vec3( gl_ModelViewMatrix * gl_Vertex );
       distance = length( listenPos.xyz - soundPos.xyz );

       testVector = normalize(position - vec3( gl_ModelViewMatrix * vec4( listenPos, 1.0 ) )
);
}
```

A.2 Fragment shader

```
/*
Brent Cowan & Bill Kapralos, June 2009

This shader samples many points between the sound source and listener gathering information
much like a person taking a survey. The data collected is rendered as an image to be read back
from the frame buffer. The image is then filtered (scaled) on the CPU in order to estimate the
occlusion, diffraction, and reverberation present in the sampled space.
*/

//objects closer than this distance could block the camera, the occlusion properties of the
//object are reduced if the distance is below the blockDist variable set by the application.
uniform float blockDist;

varying vec3  normal;     //Stores the interpolated surface normal
varying vec3  position;   //position of the object’s surface currently being rendered
varying float distance;   //distance between the sound and listener
varying vec3  testVector; //a normalized vector pointing from the listener to the point on
                          //the object currently being rendered
void main ()
{
       //The distance between the sound source and the point on the object currently being
        //rendered
       float objectDist = length(position);

       //Discard fragment if point is farther than the listener. This effectively causes the
      //view frustum to have a rounded bottom
       if(objectDist < distance )
       {
              vec3  norm    = normalize(normal); //Object’s surface normal
       //We know that the sound source is in front of the surface because the scene
       //is being rendered from the sound source’s position. If the surface normal
       //where facing away from the sound source, than the surface would not be
       //rendered due to back face culling. If the listener is also in front of the
       //surface than both the sound and listener are on the same side of the surface
       //meaning the surface is not blocking the sound. The dot product of the surface
       //normal and the test vector would return a negative number when the listener
       //is in front of the surface. This negative number would be replaced by 0 after
       //the max function has been called indicating that this surface is not
       //occluding.
       float block = max(dot(norm, normalize(testVector)), 0.0);

       //block is reduced for objects too close to the source.
       block = block * min(objectDist / blockDist, 1.0);

       //The scene begins as a white canvas. Each object rendered has its colors
       //subtracted so that overlapping objects darken the scene.
       gl_FragColor  = vec4( 0, block, 0, 1.0 );
}
else
{
       gl_FragColor = vec4( 0.0, 0.0, 0.0, 1.0 ); //Do not darken the scene
}
}
```

A.3 Receiver scaling filters

```
//Create the scaling filters
int width = 128;
int height = 64;
float *lowFilter  = new float[width*height];
float *highFilter = new float[width*height];
Vector3D v1(float(width)*0.5f, float(height), 0.0f);
Vector3D v2;
for(GLuint w=0; w<width; w++)
{
        for(GLuint h=0; h<height; h++)
        {
                v2.x = float(w); testV.y = float(h*2);
                v2 = v2-v1;
                float_A = v2.GetLength() / (float(width)*0.7071068f);

                //Create the Low Freq. Sin scaling filter
                lowFilter[w + h*width] = cos(float_A*90.0f*DTR);

                //Create the High Freq. Sin scaling filter
                highFilter[w + h*width] = pow(cos(float_A*90.0f*DTR),10.0f);
        }
}
```

References

Allen JB, Berkley DA (1979) Image method for efficiently simulating small-room acoustics. J Acoust Soc Am 65(4):943–950

Bass HE, Sutherland LC, Zuckerwar AJ (1990) Atmospheric absorption of sound: update. J Acoust Soc Am 88(4):2019–2021

Biot MA, Tolstoy I (1957) Formulation of wave propagation in infinite media by normal coordinates with an application to diffraction. J Acoust Soc Am 29(3):381–391

Buck I, Foley T, Horn D, Sugerman J, Fatahalian K, Houston M, Hanrahan P (2004) Brook for GPUs: Stream computing on graphics hardware. ACM Transactions on Graphics 23(3):777–786

Calamia PT, Svensson UP, Funkhouser TA (2005) Integration of edge-diffraction calculations and geometrical-acoustics modeling. In: Proceedings of forum acusticum 2005 (Budapest, Hungary), August 29–September 2 2005, pp 2499–2504

Cremer L, Müller HA (1978) Principles and applications of room acoustics, vol 1. Applied Science Publishers LTD, Barking

IEEE (1987) IEEE standard for binary floating-point arithmetic. ACMSIGPLAN Notices 22, 2 (Feb.), 9–25

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

Foley D, van Dam J,A, Feiner SK, Hughes JF, Phillips RL (1994) Introduction to computer graphics. Addison-Wesley Publishing Co., Reading

Funkhouser T, Tsingos N, Carlbom I, Elko G, Sondhi M, West JE, Pingali G, Min P, Ngan A (2004) A beam tracing method for interactive architectural acoustics. J Acoust Soc Am 115(2):739–756

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

Hamidi F, Kapralos B (2009) A review of spatial sound for virtual environments and games with graphics processing units. Open Virtual Real J 1(1):8–17

Hecht E (2002) Optics, 4th edn. Pearson Education Inc, San Francisco

Hillesland KE, Lastra A (2004) GPU floating-point paranoia. In: Proceedings of ACM workshop on general purpose computing on graphics processors 2004 (Los Angeles, California, USA), August 7–8 2004, C, 8

Jensen HW (2001) Realistic image synthesis using photon mapping. A. K. Peters, Ltd., Natick

Kapralos B, Jenkin M, Milios E (2008a) Sonel mapping: a probabilistic acoustical modeling method. Building Acoust 15(4):289–313

Kapralos B, Jenkin M, Milios E (2008b) Virtual audio systems. Presence Teleoperators Virtual Environ 17(6):524–549

Keller JB (1962) Geometrical theory of diffraction. J Opt Soc Am 52(2):116–130

Kuttrff H (2000) Room acoustics, 4th edn. Spon Press, London

Lokki T, Svensson P, Savioja L. An efficient auralization of edge diffraction. In: Proceedings of the audio engineering society 21st international conference on architectural acoustics and sound reinforcement (St. Petersburg, Russia), June 1–3 2002, pp 317–325

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

Mark WR, Glanville PS, Akeley K, Kilgard MJ. 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

Mehta M, Johnson J, Rocafort J (1999) Architectural acoustics principles and design. Prentice Hall Inc., Upper Saddle River

Menshikov (2003) Modern audio technologies in games. http://ixbtlabs.com/articles2/sound-technology/index.html

Murphy DT, Beeson M (2003) Modelling spatial sound occlusion and diffraction effects using the digital waveguide mesh. In: Proceedings of the audio engineering society 24th international conference: multichannel audio the new reality, (Banff, Alberta, Canada), June 26–28, 2003, pp 207–216

Owens JD, Luebke D, Govindaraju N, Harris M, Krüger J, Lefohn AE, Purcell TJ (2007) A survey of general-purpose computation on graphics hardware. Comput Graph Forum 26(1):80–113

Rost R (2006) OpenGL shading language, 2nd edn. Addison-Wesley Professional, Boston

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

Svensson UP, Fred RI, Vaderkooy J (1999) An analytic secondary source model of edge diffraction impulse responses. J Acoust Soc Am 106(5):2331–2344

Torres RR, Svensson P, Kleiner M (2001) Computation of edge diffraction for more accurate room acoustics auralization. J Acoust Soc Am 109(2):600–610

Tsingos N, Gascuel JD (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, Gascuel JD (1998) Fast rendering of sound occlusion and diffraction effects for virtual acoustic environments. In: 104th convention of the Audio Engineering Society (Amsterdam, The Netherlands), May 16–19, 1998, preprint no. 4699

Tsingos N, Funkhouser T, Ngan A, 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), 2001, pp 545–552

Tsingos N, Carlbom I, Elko G, Funkhouser T, Kubli B (2002) Validation of acoustical simulations in the "Bell Labs Box". IEEE Comput Graph Appl 22(4):28–37

Tsingos N, Dachsbacher C, Lefebvre S, Dellepiane M (2007) Instant sound scattering. Rendering Techniques. In: Proceedings of the eurographics symposium on rendering, Grenoble, France, June 25–27