Multimed Tools Appl (2016) 75:5205–5221
DOI 10.1007/s11042-015-2943-4

GPU-based image method for room impulse response calculation

Zhong-hua Fu[1] · Jian-wei Li[1]

Received: 21 April 2015 / Revised: 30 July 2015 / Accepted: 8 September 2015 /
Published online: 19 October 2015

Abstract Room impulse response (RIR) simulation based on the image-source method is widely used in room acoustic research. The calculation of the RIR in computer has to digitalize sound propagation delay into discrete samples. To carefully consider the digitalization error greatly increases the massive computational load of the image-source method. Therefore many real-time audio applications simply round-off the propagation delay to its nearest sample. This approximation, however, especially when the sampling frequency is low, degrades the phase precision that is required by applications such as microphone array. In this paper, by involving a Hanning-windowed ideal low-pass filter to reduce the digitalization error, a more precise image-source model is studied. We analyze its parallel calculation procedure and propose to use Graphics Processing Unit (GPU) to accelerate the calculation speed. The calculation procedure is divided into many parallel threads and arranged according the GPU architecture and its optimization criteria. We evaluate the calculation speeds of different RIRs using a general 5-core CPU, an ordinary GPU (GTX750) and an advanced GPU (K20C). The results show that, with similar precise RIR results, the speedup ratios of GTX750 and K20C over the general CPU can achieve 20 and 120 respectively.

Keywords Room impulse response · Graphics processing unit · Image-source model

1 Introduction

Room impulse response (RIR), being central to both measurement and modeling of room acoustics, is a very important description on the sound propagation from one point to another

✉ Jian-wei Li
ljjianweiyx@gmail.com

Zhong-hua Fu
840318140@qq.com; mailfzh@nwpu.edu.cn

[1] School of Computer Science, Northwestern Polytechnical University, Xi'an, China

point inside a room. It is widely used in acoustic signal processing, such as immersive audio communication [9], multi-input-multi-output (MIMO) audio system [10], auralization [25], etc. However, due to the underlying complexity of sound propagation, the acoustic model of RIR is very computation-intensive.

In the acoustic signal processing community, the image-source model [1] might be the most commonly used RIR modeling technique, because it covers a wide range of audio frequencies than the wave-based model like Finite Element Method (FEM), Boundary Element Method (BEM), Finite Difference Time-Domain (FDTD) methods [24, 29], etc. It is a fundamental ray-based modeling technique, which assumes that sound propagates in straight line like ray and all reflections are specular. Since it is guaranteed to find all the reflection paths, it is more accurate than other ray-based models, such as the ray-tracing [13]. However, as the reflection order increasing, the number of image sound sources boosts exponentially. Thus to consider each propagation path with respect to each image sound source requires huge computational load. While the dramatic evolution and wide applications of GPU provides another promising solution.

Over the past decade, the computational power of GPUs has been exploited for scientific, database, geometric, imaging, as well as acoustic signal processing [5, 31]. In fact, there is a growing trend in the development of computer processor. The main effort in terms of improving the calculation performance turns from making a single core faster to involving multiple cores on the same chip, such as the evolutions of multi-core CPUs and general purpose GPUs (GPGPUs). Especially the GPGPUs, since they have hundreds or thousands of simpler cores, is very suitable for higher parallel calculation tasks. In the early days, the programming on GPU was performed using graphics APIs (application programming interface) such as OpenGL and DirectX, so general calculation had to be posed as a graphics rasterization problem [11]. Later NVIDIA introduced CUDA (compute unified device architecture) in 2006 [4], which accelerates the widely applications of GPU.

The architectural characteristics of GPUs constrain their power only on numerous independent tasks that can be implemented in parallel, i.e. so-called embarrassingly parallel problems. In fact, many audio applications can be reformulated into the embarrassingly parallel problems. For example, in [6, 26], GPU is used for headphone-based sound spatialization. In [34], a GPU-based sound synthesis is proposed. In [11, 21], GPU is used for the acoustic ray-tracing modeling. There also are many works on numerical acoustics using GPU to solve FEM, BEM or FDTD problems [19, 20, 22]. Please refer to the references [25, 31] for overview on use of GPU on auralization and correlated audio processing topics.

In this paper, we focus on the RIR calculation based on the image-source method, which can be a tool for room acoustics simulation and measurement. In most of previous GPU correlated audio processing works, the major purpose is to deal with real-time audio interaction and simulate realistic auditory feelings. Thus different kinds of fast or approximate methods can be involved. For example, in ray-tracing methods or beam-tracing method only a finite number of rays are considered to be emitted by every image sound source [15, 30]. In real-time auralization, especially in dynamic environments, the approximation in geometric accuracy [27] and late reverberation [2] are used. The similar approximations are involved in non-GPU fast RIR simulations. In [8], the ray-tracing method is used for binaural RIR simulation, and in [16, 28], the precision is only preserved in the early reflections. One of the most frequently used approximation while simulating the discrete version of the RIR using a computer is to round-off the propagation delay to its nearest sample, for example, in both the original image-source method [1] and its newly fast implementation in [17]. These approximations might be ignored in many applications, however, for multiple microphones systems that are sensitive to inter-microphone phase, accurately simulate the propagation delays is critical, especially when the sampling frequency is low. A more precise RIR calculation model is proposed in [7, 18], where a Hanning-windowed ideal low-pass filter is involved to significantly reduce the round-off distortion. However, the impulse response of the low-pass filter is convolved with every image source, thus significantly increases the original massive computational load of the image-source method, especially when computing a large number of RIRs, e.g., as in the case of moving sound sources and microphone array applications.

In fact, the image-source method, even with the low-pass filter, is intuitively suit for GPU processing. Since if the room acoustic hierarchy, the locations of sound sources and receivers, are given, all of the numerous image sources can be traced independent and thus can be processed in parallel. On the other hand, GPUs currently are almost embedded in every computational platform. It is worth utilizing their power to facilitate the RIR calculation. Therefore in this paper, we study the precise image-based RIR calculation, which takes the Hanning-windowed ideal low-pass filter into consideration. The calculation procedure is reprogrammed and efficiently implemented in GPU. By optimizing thread deployment, the calculation speed with GPU is significantly improved comparing to that with CPU. The results are verified using two different GPUs of NVIDIA, GeForce GTX750 and Tesla K20C. Comparing to a normal CPU Core i5-3470 with 4 cores, the speedup ratios of GTX750 and K20C can achieve 20 and 120 respectively.

2 Image-source model and calculation analysis

In this section, we analyze the image-source model and its calculation procedure. Then the precise digitalization is considered and the bottleneck calculations are analyzed.

2.1 Image-source model

The image-source model assumes that sound propagates in straight line and all reflections are specular. Thus given the location and the normal direction of a wall, a virtual sound source, i.e. an image source, is determined by the locations of original sound source. When a single sound source plays in a reverberant room, the reverberant sound signal recorded by a receiver can be viewed as the summation of all direct sounds emitted from all image sources, which simultaneously plays the same signal as the original sound source.

In this paper, we consider a typical shoebox-shaped room, which is widely used in the context of speech reverberation, such as in [3, 7] and [17]. Since the room is in 3 dimensions, the image sources are extended in 3 dimensions space. We denote the 3 dimensions as x, y, and z, respectively. Let's suppose the size of the room is $L_x \times L_y \times L_z$, and the origin point is located at $(0, 0, 0)$. To make a clearer illustration, we first consider the wall reflection in x dimension. The reflections in 3 dimensions are the permutation of the basic case.

Suppose a sound source $\tilde{s}_0$ is located at $(\tilde{x}_0, \tilde{y}_0, \tilde{z}_0)$, and a receiver s_0 is located at (x_0, y_0, z_0), given two parallel walls located at $x = 0$ and $x = L_x$ respectively, the wall reflections in x dimension is shown in Fig. 1. Note that the sound reflection coefficients of the two walls are denoted as β_{x_1} and β_{x_2}, respectively.

Since the x-coordinate of the source $\tilde{s}_0$ is $\tilde{x}_0$, with the two walls, the x-coordinates of all sources (including image sources and the original source) in x dimension are $[\cdots, -2L_x - \tilde{x}_0, -2L_x + \tilde{x}_0, -\tilde{x}_0, \tilde{x}_0, 2L_x - \tilde{x}_0, 2L_x + \tilde{x}_0, 4L_x - \tilde{x}_0, \cdots]$. The corresponding refection orders are $[\cdots, -3, -2, -1, 0, 1, 2, 3, \cdots]$. Then the x-coordinates of the vectors pointing

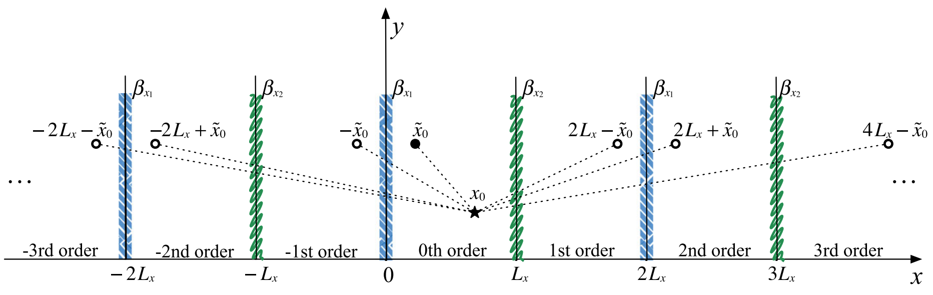

Fig. 1 The image-source model with reflections only in x dimension

from every source to the receiver are $[\cdots, x_0+\tilde{x}_0+2L_x, x_0-\tilde{x}_0+2L_x, x_0+\tilde{x}_0, x_0-\tilde{x}_0, x_0+\tilde{x}_0-2L_x, x_0-\tilde{x}_0-2L_x, x_0+\tilde{x}_0-4L_x, \cdots]$, which can be expressed in a general form as

$$(x_0-\tilde{x}_0+2p_x\tilde{x}_0)+(2m_xL_x),$$

where $p_x=1$ if the reflection order is odd, otherwise $p_x=0$, and $m_x\in Z$.

The accumulated reflection coefficients of sound signal propagated from every source to the receiver are $[\cdots, \beta_{x_1}^2\beta_{x_2}, \beta_{x_1}\beta_{x_2}, \beta_{x_1}, 1, \beta_{x_2}, \beta_{x_1}\beta_{x_2}, \beta_{x_1}\beta_{x_2}^2, \cdots]$, which also can be expressed in a general form as

$$\beta_{x_1}^{|m_x+p_x|}\beta_{x_2}^{|m_x|}.$$

The corresponding reflection order can be expressed as $N_x=|2m_x+p_x|$.

Now with the above general forms, the reflection relationship can be extended to 3-dimension space. The vector from source $\tilde{s}_k$ to the receiver s_0 is

$$\mathbf{d}_{\tilde{s}_ks_0}=\begin{bmatrix}(x_0-\tilde{x}_0+2p_x\tilde{x}_0)+(2m_xL_x)\\(y_0-\tilde{y}_0+2p_x\tilde{y}_0)+\left(2m_yL_y\right)\\(z_0-\tilde{z}_0+2p_z\tilde{z}_0)+(2m_zL_z)\end{bmatrix}, \tag{1}$$

where p_y and p_z are similar to p_x, and $m_y, m_z\in Z$. The subscript k denotes the k-th image source. Apparently, any group of values of p_x, p_y, p_z, m_x, m_y, and m_z determines one sound source. Therefore, to consider all sound sources, one needs to traverse all possible values of $p_x, \cdots, m_z$. That is the most time-consuming process. We will discuss its parallel implementation in Section 3. Accordingly, the accumulated reflection coefficient referring to the source $\tilde{s}_k$ is

$$\beta_{\tilde{s}_ks_0}=\beta_{x_1}^{|m_x+p_x|}\beta_{x_2}^{|m_x|}\beta_{y_1}^{|m_y+p_y|}\beta_{y_2}^{|m_y|}\beta_{z_1}^{|m_z+p_z|}\beta_{z_2}^{|m_z|}, \tag{2}$$

where β_{y_1} and β_{y_2} are the reflection coefficients of the walls located at $y=0$ and $y=L_y$, respectively. β_{z_1} and β_{z_2} are defined similarly. Given a group of values of $p_x, \cdots, m_z$, the reflection order of the source $\tilde{s}_k$ is

$$N=N_x+N_y+N_z=|2m_x+p_x|+\left|2m_y+p_y\right|+|2m_z+p_z| \tag{3}$$

Now the impulse response from the source $\tilde{s}_k$ to the receiver s_0 can be written as

$$h\left(\tilde{s}_k, s_0, t\right)=\alpha_{\tilde{s}_ks_0}\delta\left(t-\tau_{\tilde{s}_ks_0}\right), \tag{4}$$

where $\alpha_{\tilde{s}_ks_0}$ denotes the decay coefficient of the source $\tilde{s}_k$ and $\tau_{\tilde{s}_ks_0}$ is the corresponding propagation delay time. $\alpha_{\tilde{s}_ks_0}$ is calculated as

$$\alpha_{\tilde{s}_ks_0}=\frac{\beta_{\tilde{s}_ks_0}}{4\pi\left|\mathbf{d}_{\tilde{s}_ks_0}\right|}, \tag{5}$$

where $\left|\mathbf{d}_{\tilde{s}_k s_0}\right|$ represent the distance between the k-th source to the receiver. The propagation $\tau_{\tilde{s}_k s_0}$ can be obtained by

$$\tau_{\tilde{s}_k s_0} = \left|\mathbf{d}_{\tilde{s}_k s_0}\right| / c, \tag{6}$$

where c is the speed of sound.

Finally, the RIR from the original source to the receiver is the summation of the impulse responses of all image sources, which is

$$h\left(\tilde{\mathbf{s}}, s_0, t\right) = \sum_k h\left(\tilde{s}_k, s_0, t\right) = \sum_k \alpha_{\tilde{s}_k s_0} \delta\left(t - \tau_{\tilde{s}_k s_0}\right). \tag{7}$$

Note that to traverse all k values is equivalent to traverse all possible values of p_x, p_y, p_z, m_x, m_y, and m_z, respectively.

2.2 Discrete calculation of the impulse response

In the previous section, the theoretical RIR based on image-source geometric model is derived. To implement the calculation in computer, there are several issues needed to be considered: 1) how many image sound sources are needed for the summation in (7); 2) the time delay $\tau_{\tilde{s}_k s_0}$ in (7) may not fall at sampling instants, so the discrete version of this formula is required.

2.2.1 Number of image sources

The first issue, according to the general form in (1), is actually meant to determine the required ranges of m_x, m_y, and m_z, respectively. Generally, we suppose $-\hat{N}_u \leq m_u \leq \hat{N}_u$, where $u \in \{x, y, z\}$. If $\hat{N}_u$ is defined, then the number of image sources are known. Note that $\hat{N}_u$ is not strictly equal to the reflection order, i.e. N_u, in u direction.

In practice, $\hat{N}_u$ can be either user defined or determined according to the so-called reverberation time $T_{60}(f)$, which is frequency dependent and determined by the room geometry and the sound absorption by all materials in the room. There are tremendous works in the literature that address the reverberation time estimation problem, [14]. In this work, we use the modified Sabine formula [12] as

$$T_{60}(f) = \frac{55.3V}{c\left[A_{\text{mat}}(f) + 4m_{\text{air}}(f)V\right]}, \tag{8}$$

where V is the room volume, c is sound speed, $A_{\text{mat}}(f)$ is the total material absorption surface, and $m_{\text{air}}(f)$ is the sound intensity absorption coefficient of air.

Supposing the air in the room is steady, the sound speed can be obtained as

$$c = 331\sqrt{1 + 0.0036T}, \tag{9}$$

where T is the air temperature in Celsius. The total material absorption surface $A_{\text{mat}}(f)$ is calculated as

$$A_{\text{mat}}(f) = \sum_{i=1}^{6}\left(1 - {\beta_i}^2\right) S_i, \tag{10}$$

where $\beta_i(f)$ is the frequency-dependent reflection coefficient of i-th wall, and S_i is the wall surface. The sound intensity absorption coefficient of air, $m_{\text{air}}(f)$, can be estimated using

$$m_{\text{air}}(f) = 5.5 \times 10^{-4} \times \left(\frac{50}{H}\right) \times \left(\frac{f}{1000}\right)^{1.7}, \tag{11}$$

where H is the humidity of air.

Now with the frequency-dependent $T_{60}(f)$, $\hat{N}_u$ can be determined by

$$\hat{N}_u = \frac{c\left\lceil \max_f T_{60}(f) \right\rceil}{2L_u}, \tag{12}$$

where L_u is the room size in u dimension. Consequently, with $p_u \in \{0, 1\}$, and $-\hat{N}_u \leq m_u \leq \hat{N}_u$, where $u \in \{x, y, z\}$, to traverse all image sources, there are

$$\hat{N} = 2^3(2\hat{N}_x + 1)(2\hat{N}_y + 1)(2\hat{N}_z + 1) \tag{13}$$

image sources are involved for the RIR calculation.

If the sampling frequency is f_s, the length of the final RIR is

$$L_{\text{RIR}} = f_s \max_f T_{60}(f). \tag{14}$$

2.2.2 *Discrete form of the impulse response*

The second issue is to turn the impulse $\delta\left(t - \tau_{\tilde{s}_k s_0}\right)$ in (4) into its discrete form $\delta\left(n - \tau_{\tilde{s}_k s_0} f_s\right)$ so that the impulse responses of all image sources can be accumulated (7) in computer. The problem is that $\tau_{\tilde{s}_k s_0} f_s$ may not be integer.

The simplest method is to fix $\tau_{\tilde{s}_k s_0} f_s$ to the nearest integer value [1]. This approximation may not influence many applications such as auralization, but for multiple microphones systems that are sensitive to arrival time, more critical solution is necessary. We follow the method proposed in [7, 18] by replacing $\delta(t)$ with the impulse response of a Hanning-windowed ideal low-pass filter as

$$\delta_{\text{LPF}}(t) = \begin{cases} \frac{1}{2}\left(1 + \cos\left(\frac{2\pi t}{T_\omega}\right)\right) \text{sinc}\left(2\pi f_c t\right) & -\frac{T_\omega}{2} < t < \frac{T_\omega}{2} \\ 0 & \text{otherwise} \end{cases}, \tag{15}$$

where T_ω is the width (in time) of the impulse response, and f_c is the cut-off frequency. As in [7], the typical value of T_ω is 8ms and f_c is set to the Nyquist frequency. The discrete form of $\delta_{\text{LPF}}(t)$ is then

$$\delta_{\text{LPF}}(n-\varepsilon) = \begin{cases} \frac{1}{2}\left(1 + \cos\left(\frac{2\pi(n-\varepsilon)}{N_\omega}\right)\right) \text{sinc}\left(2\pi f_c'(n-\varepsilon)\right) & -\frac{N_\omega}{2} < n < \frac{N_\omega}{2} \\ 0 & \text{otherwise} \end{cases}, \tag{16}$$

where $N_\omega = T_\omega f_s$ is the window width (in samples), $f_c' = f_c \big/ f_s$ is the normalized cut-off frequency, and $\varepsilon = \tau_{\tilde{s}_k s_0} f_s - \text{floor}\left(\tau_{\tilde{s}_k s_0} f_s\right)$. floor$(\cdot)$ means round towards minus infinity, so $0 \leq \varepsilon < 1$.

The above replacement of $\delta(t)$ with $\delta_{\text{LPF}}(n-\varepsilon)$ modifies the calculation of the RIR. With $\delta(t)$, each image sound source only contributes to one point in the continuous RIR, while with $\delta_{\text{LPF}}(n-\varepsilon)$, each source will contribute to N_ω samples in the discrete RIR, which requires another iteration of calculation.

2.3 Calculation analysis

According to the above analysis, the calculation procedure of the RIR consists of:

(1) Calculating the sound speed c in a given environment (9);
(2) Calculating the reverberation time $T_{60}(f)$ (8);
(3) Calculating the length of the RIR L_{RIR} (14);
(4) Calculating the $\hat{N}_u$ to determine the number of image sources $\hat{N}$ (12) and (13);

(5) Calculating the propagation delay $\tau_{\tilde{s}_k s_0}$ and the decay coefficient $\alpha_{\tilde{s}_k s_0}$ of each image sound source (1) and (2);
(6) Overlapping and adding $\delta_{\mathrm{LPF}}(n - \varepsilon)$ of each source to the final RIR (7).

One can find that in the above algorithm, the first four steps are identical for every image source. So they need to be implemented only once. However, the last two steps have to be implemented for every image source. If the calculations are implemented in sequence to traverse all image sources, it will be too time-consuming to be applied in many applications. For example, with a cube room of size 5m $\times$ 5m $\times$ 5m and 0.3s reverberation time, then $\hat{N}_u \simeq 10$, and the number of images sound source is about 74088. Suppose the sampling frequency $f_s = 44100$Hz, then the total length of the RIR is 13230, and the lowpass filter length $N_\omega = 353$. Thus there are $74088 \times 353 \simeq 2.6 \times 10^7$ points need to be overlapped and added to an array of 13230. This number of iterations is truly huge and the calculation in sequence is very inefficient and time-consuming. In fact, it is easy to find that when the locations of all sources are known, the succeeding calculations of the impulse response with respect to each source are independent, which fits the requirement of parallel computing very well.

3 GPU implemetation of the image-source model

3.1 Some optimization principles for GPU programming

In order to improve the calculation speed of an algorithm with GPU, it is necessary to use GPU efficiently. The most important thing is to organize the available resources of GPU properly. When the GPU resource is well organized, CPU can launch a kernel function to GPU to start computing [32].

The first issue is the memory resource in GPU, since the speed of accessing different kind memory varies a lot. The memory resource in GPU can be divided into on-chip memory and off-chip memory, and the available memory types include:

- Shared on-chip memory, which is shared by all threads within the same thread-block;
- Global memory, which is the biggest off-chip memory and has much smaller bandwidth than on-chip memory;
- Constant memory, which is read-only off-chip memory and can be accessed faster then the Global memory due to its constant cache;
- Texture memory, which is read-only off-chip memory with texture cache.
- Local memory, which is also off-chip memory without cache, and can be occupied by only one thread privately.

Generally, the global memory, the local memory and the texture memory have longer access delay than the constant memory, and the on-chip shared memory is the fastest memory that should be used preferably. Currently, the widely used NVIDIA GPU based on Kepler architecture has 16KB local memory, 64KB constant memory and 16KB shared memory. In addition to those memory resources, every streaming multiprocessor has 8192 or 16384 32-bits registers.

The second issue is the organization of threads. There are hundreds or thousands of threads running on a GPU simultaneously. In NVIDIA programming model, it allows 32 threads to be organized as one warp, and the threads within one warp are executed parallel. It also allows several warps to compose as one block, and the threads within one block are executed in one streaming multiprocessor. This facilitates the communication between different threads because: (1) the threads within one block share same memory; (2) it is easy to synchronize the threads in one block. Note that the maximum amount of threads allowed within one block is limited. For threads in different block, the communication must go through the global memory, which will slow down the calculation speed. The general optimization principles of GPU programming can be summarized as [23, 32]:

- More threads are better, so as to deemphasize the memory access delay;
- Avoid access of global memory;
- Try to organize threads within one block. Note the number of threads within one block should be an integer multiple of the number within one warp;
- Try to reduce the communication between device and host to avoid long delay.
- It is better to use alignment access in the global memory. [23, 33]

3.2 Implementation on GPU

Recalling the calculation procedure of the RIR in Section 2.3, the first four steps are to be implemented only once. So they are performed in CPU in sequence. When these steps are done, to traverse all image sound sources as fast as possible, the most frequently access variables, including the counters, i.e. p_x, p_y, p_z, m_x, m_y, m_z, the coordinates of the sound source $(\hat{x}_0, \hat{y}_0, \hat{z}_0)$ and the receiver (x_0, y_0, z_0), the room size (L_x, L_y, L_z), the reflection coefficients of the six walls, $\beta_{x_1}, \cdots, \beta_{z_2}$, the sound speed c, the length of the RIR L_{RIR}, $\hat{N}_x$, $\hat{N}_y$, $\hat{N}_z$, the sampling frequency f_s, and the lowpass filter coefficients $\delta_{\text{LPF}}(n)$ are stored in the GPU registers. These variables will be accessed frequently in the calculations of the step (5) and step (6). Next we consider the thread organization in GPU for these two steps. To save the result RIR data, a buffer of length L_{RIR} is allocated in the global memory.

3.2.1 Calculating the propagation delay and decay coefficient

Since the number of all image sources $\hat{N} = 2^3(2\hat{N}_x + 1)(2\hat{N}_y + 1)(2\hat{N}_z + 1)$, according to the first GPU optimization principle list in Section 3.1, it is best to perform $\hat{N}$ threads in parallel. And in the CUDA programming model framework, it is suggested to deploy all threads into one block. However, the NVIDIA GPU only supports at most 1024 threads in one block, which is generally less than what we need. Therefore, we have to organize more blocks and maintain the number of threads within one block as many as possible.

According to the CUDA programming model, the blocks can be arranged in one-dimension, two-dimension, or three-dimension. The threads within each block can also be arranged in one-, two-, or three-dimension. To address a thread within one block, CUDA provides two predefined structures named as $blockIdx$ and $threadIdx$. Each structure has three members, $.x$, $.y$, and $.z$, referring to the indexes in x-dimension, y-dimension and z-dimension, respectively. In our task, to make use of these predefined counters and deploy more threads within each block, we arrange a two-dimension block array, and the threads within each block is arranged in one-dimension. The thread organization for propagation delay and decay coefficient calculations is shown in Fig. 2, where $B_x = 2\hat{N}_x + 1$, $B_y = 2\hat{N}_y + 1$, and $B_z = 2\hat{N}_z + 1$. Therefore, to address a thread, we need three counters. Two of them address the corresponding block, and one of them addresses the target thread.

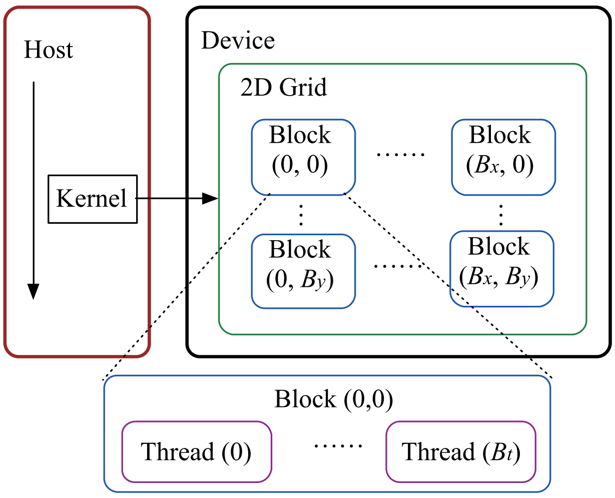

Fig. 2 The thread organization in GPU for propagation delay $\tau_{\tilde{s}_k s_0}$ and decay coefficient $\alpha_{\tilde{s}_k s_0}$ calculations. The blocks are organized into a two-dimension matrix of $B_x \times B_y$, and within each block, there are B_t threads

The three counters, as well as their mapping to the counters of traversing all image sources, are proposed as,

$$\begin{aligned} m_x &= blockIdx.x; \\ m_y &= blockIdx.y; \\ m_z &= threadIdx.x; \end{aligned} \tag{17}$$

where $-\hat{N}_u \le m_u \le \hat{N}_u$, $(u \in \{x, y, z\})$. Within each thread, 2^3 image sources $(p_x, p_y, p_z = 0 \,\text{or}\, 1)$ are calculated in sequence. That reduces the possibility of thread switching, which also depress the calculation efficiency.

The result propagation delay $\tau_{\tilde{s}_k s_0}$ and decay coefficient $\alpha_{\tilde{s}_k s_0}$ with respect to each image source is stored respectively into a one-dimension array allocated in the global memory. The entry index of each array is determined by a group of $(p_x, p_y, p_z, m_x, m_y, m_z)$.

3.2.2 Overlap-adding to accumulate the room impulse response

The basic consideration in organization GPU resource for this calculation is similar as the above step. However, because of the low-pass filter window, now we have to arrange more threads. Specifically, for each image source, there are N_ω samples need to be calculated and accumulated into the final RIR. Therefore, we have

$$N_\omega \cdot 2^3(2\hat{N}_x + 1)(2\hat{N}_y + 1)(2\hat{N}_z + 1)$$

threads in total.

Generally, $N_\omega > \max\left(2\hat{N}_x + 1, 2\hat{N}_y + 1, 2\hat{N}_z + 1\right)$. i.e. the length of the low-pass filter window is longer than the largest reflection order. Additionally, sometimes $2^3 \cdot \min\left(2\hat{N}_x + 1, 2\hat{N}_y + 1, 2\hat{N}_z + 1\right) > 1024$. Therefore we have to arrange a three-dimension block matrix, the size of which is $(2\hat{N}_x + 1)$ by $(2\hat{N}_y + 1)$ by $2^3(2\hat{N}_z + 1)$, where each block refers to one image source. And within each block, there are (N_ω) threads. The arrangement is shown in Fig. 3, where

$$\begin{aligned} B_x &= 2\hat{N}_x + 1, \\ B_y &= 2\hat{N}_y + 1, \\ B_z &= 2^3(2\hat{N}_z + 1), \\ B_t &= N_\omega. \end{aligned}$$

When the low-pass filter window of each image source is weighted by the decay coefficient $\alpha_{\tilde{s}_k s_0}$, it is moved to the offset of $\tau_{\tilde{s}_k s_0} f_s$ and accumulated into the final RIR. Note that $\tau_{\tilde{s}_k s_0} f_s$ must be rounded towards nearest integer firstly.

There is another thing needs consideration. It can be expected that some virtual sound sources may have same propagation delay, so the threads correlated to those sources will access same offset in the final RIR cache. If multiple threads operate same cache unit simultaneously, access violation may result in failure operation. To avoid this problem, we use the atomic operation, which is a common operation that will always be executed without interruption during the operation. Although the atomic operation will extend the computation time, it is necessary for achieving correct result.

4 Experiments

The purpose of the experiments in this section consists of two parts. The first one is to validate the correctness of the proposed GPU-based calculation. The second one is to evaluate the speedup ratio of the GPU-based calculation compared to the CPU-based calculation. The baseline program is from [7]. Note that the estimation on reverberation time in that program is slightly different than ours. To make fair comparison, the reflection order or the length of impulse response in the rest of the paper is directly defined, therefore the results has no relationship with the reverberation time estimation. By changing the reflection order, the corresponding execution times are measured.

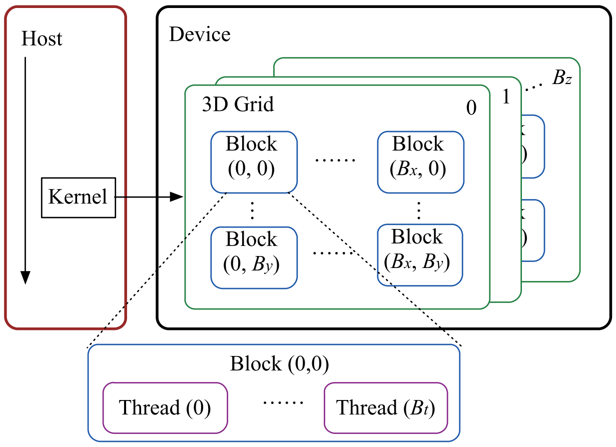

Fig. 3 The thread organization in GPU for overlap-adding to accumulate the final room impulse response. The blocks are organized into a three-dimension matrix of $B_x \times B_y \times B_z$, and within each block, there are B_t threads

4.1 Experiments setup

The baseline program and the proposed GPU-based program are implemented in the same computer. For the GPU-based program, two different GPUs from NVIDIA are involved. The details of the computation devices are list in Table 1. Note that GPU1 (denoted as K20C) is an advanced GPU, and GPU2 (denoted as GTX750) is an ordinary GPU. One can find that GPU1 has not only far more CUDA cores, but also significantly larger memory and wider bandwidth than those of GPU2. The competitor CPU is a general one with 5 cores and much higher central clock than the GPUs. We simulate 3 shoebox-shaped rooms of different sizes, which are denoted as Room1, Room2, and Room3, respectively. The details of the rooms are list in Table 2. In all of the following experiments, the sampling frequency is 44.1kHz. In order to compare the RIR calculation performance in extremely massive conditions, the wall reflection coefficients are all set to 0.9, which means there will be numerous image sources need calculation. Especially in the smallest room, Room3, due to its small size and strong reflection, the reflection order of the image-source model is significantly high.

In order to compare the calculation speeds of different devices, for each room, we defined 5 different lengths of the corresponding RIR. The first four lengths are 4096, 8192, 16384, 44100, and the last length is equal to the T_{60}(reverberation time in samples) of each room. Note that longer T_{60} doesn't refer to higher computational load. The key issue is the number of image sources. For example, Room3 has shortest reverberation time, but the number of image sources is about 2 times of Room1, the one with longest reverberation time. The RIRs are calculated using the CPU and the GPUs respectively, then the computation time are measured, so as to estimate the speedup ratio. Note that in some cases the running time of the GPUs is less than 1ms that is too small to measure. To obtain the running time precisely, the corresponding calculation is repeated 80000 times so as to get an average running time.

4.2 Experiments results

4.2.1 Calculation errors evaluation

Firstly we exam the correctness of the RIR calculation using the two GPUs in Table 1. The reference is the calculation result using the CPU. Figure 4. It is found that there are only tiny numerical errors which are due to the difference of the CPU and the GPUs in calculating several mathematic functions. It is confirmed that when closing down these functions, the

Table 1 Computing device information

Processor	CPU	GPU1	GPU2
Version	Core i5-3470	NVIDIA Tesla K20C	NVIDIA GeForce GTX750
Number of Core	4	2496 CUDA cores	640 CUDA cores
Clock	3.2GHz	706MHz	1020MHz
Memory Size	4GB	5GB	2GB
Memory Bandwidth	—	208GB/s	86.4GB/s
TFLOPS*	—	3.524	1.04

*teraFLOPS=10^{12}FLOPS

Table 2 The information of the experiment rooms

	Room1	Room2	Room3
Size	15m×20m×6m	8m×10m×3.5m	4m×5m×3.5m
T_{60}	2.8s	1.6s	1.1s
Source location		(3.5m,2.5m,1m)	
Temperature		15°	
Humidity		0.3	
Reflection coefficient		0.9 for all walls	

consequent results are then identical. The quantitative results of the calculation errors in different testing cases are list in Table 3, where the errors are measured using the normalized misalignment defined as

$$\varepsilon = 20\log_{10}\frac{\|h_{\mathrm{CPU}}(n) - h_{\mathrm{GPU}}(n)\|_2}{\|h_{\mathrm{CPU}}(n)\|_2}, \tag{18}$$

where $\|\cdot\|_2$ denotes the l_2 norm. Note that the results of the two kinds of GPUs are identical, so the normalized misalignment is calculated between the results of the GTX750 and the Core i5-3470.

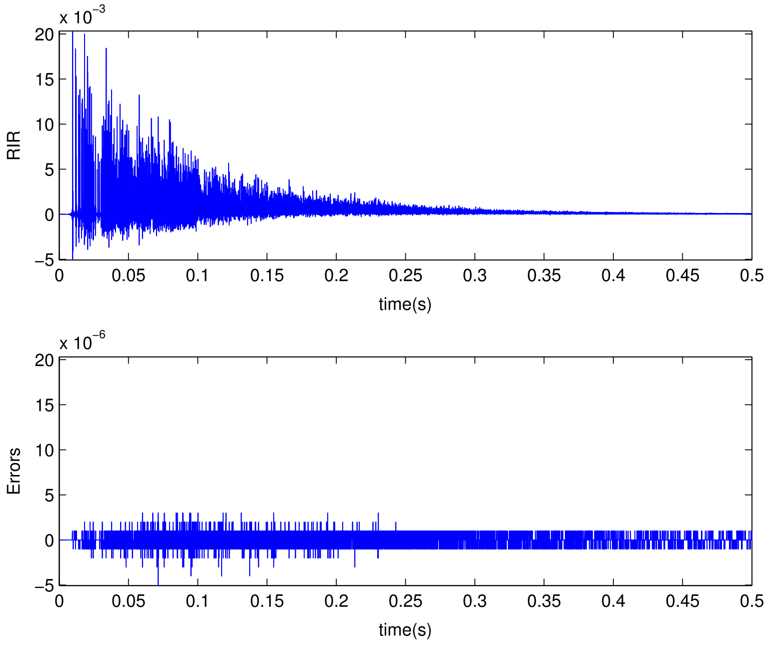

Fig. 4 The upper figure is the RIR of Room3 calculated using the CPU. The lower figure is the calculation error using GTX750 GPU

Table 3 The normalized misalignment (dB) between the RIRs using CPU and GPUs

RIR length (samples)	4096	8192	16384	44100	$T_{60} f_s$
1	−73.52	−67.00	−62.60	−58.52	−57.46
2	−70.95	−67.31	−64.25	−62.48	−62.37
3	−72.11	−69.15	−67.42	−66.93	−66.93

4.2.2 Speedup ratio evaluation

The calculation time on RIR of different lengths using the CPU and the two GPUs are list in Table 4. As expected, the calculation time of both the CPU and the GPUs are increased with the increase of the RIR length, but the time of CPU increase significantly higher than the two GPUs, and the K20C GPU gives the best results. The reason is that the number of image sound sources need calculation grows exponentially. For iterative calculation in CPU, the time is increased similarly, while for parallel calculation in GPU, due to more threads are preformed simultaneously so that the time does not increase significantly. Additionally, one can find that, given same length of the RIR, the calculation time of the small room (Room3) is longer than the large room (Room1). That is because the reflection order $\hat{N}_u$ is inversely proportional to the room size, when the wall reflection coefficients and the RIR length are same, there will be more image sources in the small room.

The speedup ratio of the GPU-based calculation is shown in Fig. 5. Each GPU results of the three rooms are averaged and compared to the CPU results. Note that since the T_{60} of the three rooms are different, the speedup ratios at $T_{60} f_s$ can not be averaged. So they are not shown. It is found that with the very general GPU, GTX750, the speedup ratios are from 8 to 21 times than the CPU, and the speedup ratios of K20C are significantly higher. We

Table 4 The calculation time comparisons

Room1 ($T_{60} = 2.8s$)						
RIR length (samples)		4096	8192	16384	44100	$T_{60} f_s$
Number of Sources		840	3640	18216	216600	3877272
	CPU	1	5	34	680	14780
Calculation time (ms)	GTX750	0.15	0.43	2.2	31	717
	K20C	0.08	0.14	0.45	21	114
Room2 ($T_{60} = 1.6s$)						
RIR length (samples)		4096	8192	16384	44100	$T_{60} f_s$
Number of Sources		275	12312	75480	1167880	4451480
	CPU	3	28	224	4374	17281
Calculation time (ms)	GTX750	0.32	1.7	16	234	795
	K20C	0.11	0.34	2	33	131
Room3 ($T_{60} = 1.1s$)						
RIR length (samples)		4096	8192	16384	44100	$T_{60} f_s$
Number of Sources		7128	38760	263736	4551240	5970600
	CPU	14	110	887	17250	23124
Calculation time (ms)	GTX750	0.94	16	47	842	1076
	K20C	0.22	2	8	135	179

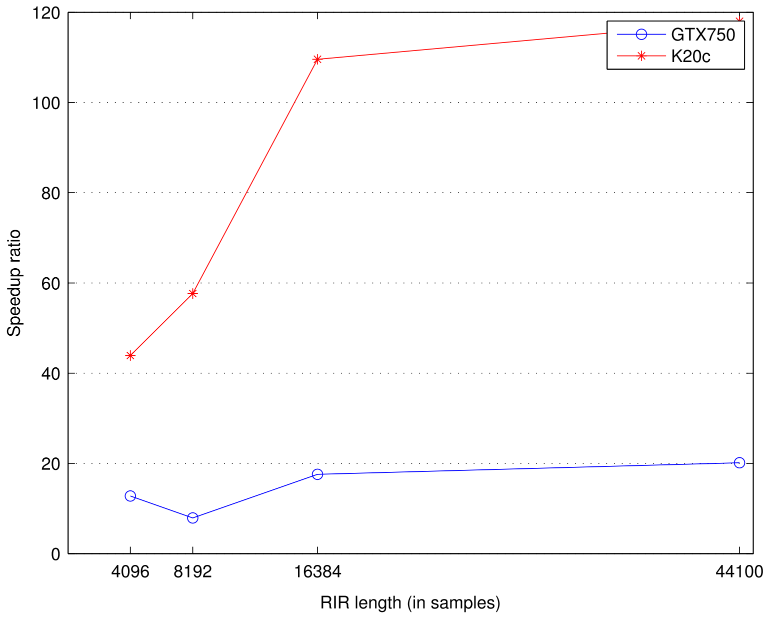

Fig. 5 The speedup ratios of the two GPUs (GTX750 and K20C) with respect to the CPU (Core i5-3470)

can infer that as the RIR length increasing, the speedup ratios of both GPUs can be further higher, but they will be constrained by the available parallel resources in GPU.

Since many real-time audio rendering applications do not require the proposed precise RIR calculation, we also present the performance comparison of the basic image-source method without the low-pass filter. It can be viewed as the computational cost spent on the RIR calculation for such real-time rendering applications. The results are shown in Table 5. Comparing to the results in Table 4, one can find that the missing of the low-pass filter significantly decreases the calculation time. For example, in Room1, with the low-pass filter, the calculation time for T_{60} on CPU is 14780ms, but without the filter, the calculation time reduces to 125ms. When using GPUs, the calculation time is also significantly decreased. Even in the worst case, i.e. in Room3, the speedup ratio of the ordinary GPU, GTX750, is about 6.5 times over the CPU. Notice that the speedup ratio of the similar case but with the low-pass filter is 21.5 (in Table 4, $23124/1076 \approx 21.5$), which means the GPUs is more powerful for the precise RIR calculation problem addressed in this paper.

Suppose that the analysis frame length is 1024 samples with 44100Hz sampling rate, the calculation time must be less than $1024/44.1 \approx 23.2$ms for real-time implementation. Obviously, in all cases but in Room3, the calculation time on GPUs satisfies the real-time requirement. It must be noted that Room3 is a small but strong reverberant room, which is

Table 5 The calculation time (ms) and speedup ratio comparisons without the low-pass filter

Cal. time (speedup ratio)	Room1	Room2	Room3
CPU	125 (-)	141 (-)	202 (-)
K20C	3 (41.7)	4 (35.3)	4 (50.5)
GTX750	16 (7.8)	16 (8.8)	31 (6.5)

seldom exist in reality. All of the test cases are designed specifically for evaluation. In fact, for real-time applications, some simplification of the image-source model must be taken into consideration to reduce the necessary computation units, which is beyond the scope of this paper.

5 Conclusions

This paper addresses the GPU-based room impulse response calculation. Specifically, a precise RIR calculation method using the image-source model is studied. To avoid the digitalization error caused by rounding off the propagation delay of each image source, a hanning-windowed ideal low-pass filter is involved. Since the overall computational load is very high but intuitively parallelizable, we use GPU programmed with CUDA to efficiently solve this problem. The bottleneck computation steps are deployed into many threads, which can be efficiently calculated in parallel with GPU. We compare the numerical errors and calculation time of different RIRs using a general 5-core CPU and two different GPUs, one advanced GPU K20C and one ordinary GPU GTX750. The results show that, with similar precise RIR results, the speedup ratios of GTX750 and K20C over the general 5-core CPU can achieve 20 and 120 respectively.

References

1. Allen JB, Berkley DA (1979) Image method for efficiently simulating small room acoustics. J Acoust Soc Am 65(4):943–950
2. Antani L, Manocha D (2013) Aural proxies and directionally-varying reverberation for interactive sound propagation in virtual environments. IEEE Trans Vis Comput Graph 19(4):567–575. doi:10.1109/TVCG.2013.27
3. Campbell DR, Palomaki KJ, Brown GJ (2005) A matlab simulation of "shoebox" room acoustics for use in research and teaching. J Comput Inf Syst (JCIS) 9(3):48–51
4. corporation N (2009) Nvidia cuda programming guide Tech. rep., NVIDIA corporation
5. Cowan B, Kapralos B (2011) Gpu-based acoustical diffraction modeling for complex virtual reality and gaming environments. In: Proceedings of the AES International Conference, pp. Binari Sonori; DOLBY; iZotope –. London, United kingdom
6. Gallo E, Tsingos N (2004) Efficient 3d audio processing on the gpu. In: ACM Workshop on General Purpose Computing on Graphics Processors. Los Angeles
7. Habets EAP (2010) Room impulse response generator. Tech. rep., International Audio Laboratories Erlanggen, Netherlands http://www.audiolabs-erlangen.de/
8. He J, Zhu M (2013) Simulation of combined head and room impulse response based on sound ray tracing in frequency domain. In: Smart and Sustainable City 2013 (ICSSC 2013), IET International Conference on, pp. 361–365. doi:10.1049/cp.2013.1957
9. Huang Y, Chen J, Benesty J (2011) Immersive audio schemes. IEEE Signal Process Mag 28(1):20–32. doi:10.1109/MSP.2010.938754
10. Huang YA, Benesty J, Chen J (2006) Acoustic MIMO Signal Processing. Springer
11. Jedrzejewski M, Marasek K (2004) Computation of room acoustics using programmable video hardware. In: International Conference on Computer Vision and Graphics (ICCVG2004). Warsaw, Poland, pp 587–592
12. Kinsier LE, Frey AR, Coppens AB, Sanders JV (2009) Fundamentals of Acoustics, 4th edn. Wiley
13. Kulowski A (1985) Algorithmic representation of the ray tracing technique. Appl Acoust 18(6):449–469

14. Kuttruff H (2000) Room Acoustics, 4edn. Taylor & Francis
15. Laine S, Siltanen S, Lokki T, Savioja L (2009) Accelerated beam tracing algorithm. Appl Acoust 70(1):172–181. doi:10.1016/j.apacoust.2007.11.011
16. Lehmann E, Johansson A (2010) Diffuse reverberation model for efficient image-source simulation of room impulse responses. IEEE Trans Audio Speech Lang Process 18(6):1429–1439. doi:10.1109/TASL.2009.2035038
17. McGovern SG (2009) Fast image method for impulse response calculations of box-shaped rooms. Appl Acoust 70:182–189
18. Peterson P (1986) Simulating the response of multiple microphones to a single acoustic source in a reverberant room. J Acoust Soc Am 80(5):1527–1529
19. Raghuvanshi N, Lloyd B, Lin M (2009) Efficient numerical acoustic simulation on graphics processors using adaptive rectangular decomposition. In: EAA Symp. on Auralization
20. Raghuvanshi N, Narain R, Lin M (2009) Efficient and accurate sound propagation using adaptive rectangular decomposition. IEEE Trans Vis Comput Graph 15(5):789–801. doi:10.1109/TVCG.2009.28
21. Rober N, Kaminski U, Masuch M (2007) Ray acoustics using computer graphics technology. In: 10th International Conference on Digital Audio Effects (DAFx-07). Bordeaux, France
22. Rober N, Spindler M, Masuch M (2006) Waveguide-based room acoustics through graphics hardware. In: International Computer Music Conference (ICMC)
23. Ryoo S, Rodrigues CI, Baghsorkhi SS, Stone SS, Kirk DB, mei W, Hwu W (2008) Optimization principles and application performance evaluation of a multithreaded gpu using cuda. In: PPoPP2008. Salt Lake City, Utah, USA
24. Savioja L (1999) Modeling techniques for virtual acoustics. Ph.D. thesis, Helsinki University of Technology. Espoo, Finland
25. Savioja L, Manocha D, Lin MC (2010) Use of gpus in room acoustic modeling and auralization. In: Proceedings of the International Symposium on Room Acoustics (ISRA), pp. 1–7. Melborne Australia
26. Siltanen S, Lokki T, Savioja L (2009) Frequency domain acoustic radiance transfer for real-time auralization. Acta Acust United Ac 95(1):106–117. doi:10.3813/AAA.918132
27. Siltanen S, Lokki T, Savioja L, Christensen CL (2008) Geometry reduction in room acoustics modeling. Acta Acust United Ac 94(3):410–418. doi:10.3813/AAA.918049
28. Svensson U, Zidan HB, Nielsen J (2011) Properties of convolved room impulse responses. In: Applications of Signal Processing to Audio and Acoustics (WASPAA), 2011 IEEE Workshop on, pp. 205–208. doi:10.1109/ASPAA.2011.6082341
29. Svensson UP, Kristiansen U (2002) Computational modelling and simulation of acoustic spaces. In: Proceedings AES 22nd Conference on Virtual, Synthetic and Entertainment Audio. Espoo, Finland, pp 11–30
30. Taylor M, Chandak A, Mo Q, Lauterbach C, Schissler C, Manocha D (2010) i-sound: Interactive gpu-based soun auralization in dynamic scenes. Technical report TR10-006, Computer Science. University of North Carolina, Chapel Hill
31. Tsingos N (2009) Using programmable graphics hardware for auralization. In: Proceedings of the EAA Symposium on Auralization. Espoo, Finland
32. Verdoolaege S, Juega JC, Cohen A, Gomez JI, Tenllado C, Catthoor F (2013) Polyhedral parallel code generation for cuda. ACM Trans Archit Code Optim 9(4):54:1–54:24
33. Yang Y, Li C, Zhou H (2015) Cuda-np: Realizing nested thread-level parallelism in gpgpu applications. J Comput Sci Technol 30(1):3–19
34. Zhang Q, Ye L, Pan Z (2005) Physically-based sound synthesis on gpus. In: Kishino F, Kitamura Y, Kato H, Nagata N (eds) Entertainment Computing - ICEC 2005, Lecture Notes in Computer Science vol. 3711, pp. 328–333. Springer, Berlin Heidelberg. doi:10.1007/11558651_32

Zhong-Hua Fu received the Ph.D. degrees from the School of Computer Science, Northwestern Polytechnical University, China, in 2004. He is currently an Associate Professor in the School of Computer Science, Northwestern Polytechnical University. His research interests are speech and audio signal processing, array speech enhancement, virtual auditory.

Jian-wei Li is a Master graduate student at the School of Computer Science, Northwestern Polytechnical University, China. His research interests are audio signal processing, virtual sound rendering and heterogeneous Development.