gpuRIR: A Python Library for Room Impulse Response Simulation with GPU Acceleration

David Diaz-Guerra, Antonio Miguel and Jose R. Beltran

Department of Electronic Engineering and Communications
University of Zaragoza
Zaragoza, Spain
{ddga, amiguel, jrbelbla}@unizar.es

Abstract—The Image Source Method (ISM) is one of the most employed techniques to calculate acoustic Room Impulse Responses (RIRs), however, its computational complexity grows fast with the reverberation time of the room and its computation time can be prohibitive for some applications where a huge number of RIRs are needed. In this paper, we present a new implementation that dramatically improves the computation speed of the ISM by using Graphic Processing Units (GPUs) to parallelize both the simulation of multiple RIRs and the computation of the images inside each RIR. We provide a Python library under GNU license that can be easily used without any knowledge about GPU programming and we show that it is about 100 times faster than other state of the art CPU libraries.

Index Terms—Room Impulse Response (RIR), Image Source Method (ISM), Room Acoustics, Graphic Processing Units (GPUs)

I. Introduction

The simulation of the acoustics of a room is needed in many fields and applications of audio engineering and acoustic signal processing, such as training robust Speech Recognition systems [1] or evaluating Sound Source Localization [2] or Speech Enhancement algorithms [3]. Although there are many low complexity techniques to simulate the reverberation effect of a room in real time, as the classic Schroeder Reverberator [4], some applications require each reflection in the reverberation to be exactly as it would be in the real room. For this type of applications, it is needed to properly simulate the Room Impulse Response (RIR) between the source and receiver positions and then filter the source signal with it.

The Image Source Method (ISM) is probably the most used technique for RIR simulation due its conceptual simplicity and its flexibility to modify parameters such as the room size, the absorption coefficients of the walls, and the source and receiver positions. We can get any level of reverberation modifying the room size and the absorption coefficients, but the computational complexity of the algorithm grows fast as the number of reflections to simulate increases. In addition, many applications require the computation of multiple RIRs for several source and receiver positions, e.g. to simulate a moving source recorded with a microphone array.

Firstly developed to support the graphics computations of video-games, Graphics Processing Units (GPUs) are today one of the best and cheapest ways to increase the speed of many algorithms that can be expressed in a parallel form.

Despite parallelizing most of the stages of the ISM is quite straightforward, to the best of our knowledge, only [5] propose to implement it in GPUs. In this paper we present a new GPU implementation with a higher degree of parallelization, better performance, and, most importantly, we provide it as a free and open-source Python library to be used by the research community[1]. Using our library does not require any knowledge about GPU programming, but just having a CUDA compatible GPU and the CUDA Toolkit and to install the library and use it as any RIR simulation library.

The reminder of this paper is structured as follows. We review the ISM in section II and section III explains how we have parallelized it. Finally, in section V, we compare the performance of our library against two of the most commonly used RIR simulation libraries and section VI concludes the paper.

II. The Image Source Method (ISM)

The image method has been widely used in many fields of physics to solve differential equations with boundary conditions, but its application for RIR estimations was originally proposed by Allen and Berkley [6]. In this section, we first review their original algorithm and then explain some of the improvements that have been proposed to improve both its accuracy and computational performance.

A. Original Allen and Berkley algorithm

The main idea behind the ISM is to compute each wavefront that arrives to the receiver from each reflection on the walls as the direct path received from an equivalent (or image) source. In order to get the positions of these image sources, we need to create a 3D grid of mirrored rooms with the reflections of the room in each dimension; as shown in Fig. 1 simplified to 2D for an example.

If the number of images we want to compute for each dimension are N_x, N_y and N_z, then we define a grid $\mathcal{N}$ of image sources $\mathbf{n} = (n_x, n_y, n_z) : -N_x/2 \leq n_x < N_x/2, -N_y/2 \leq n_y < N_y/2$ and $-N_z/2 \leq n_z < N_z/2$. The coordinates of the position of each image $\mathbf{p_n} = (x_\mathbf{n}, y_\mathbf{n}, z_\mathbf{n})$ are
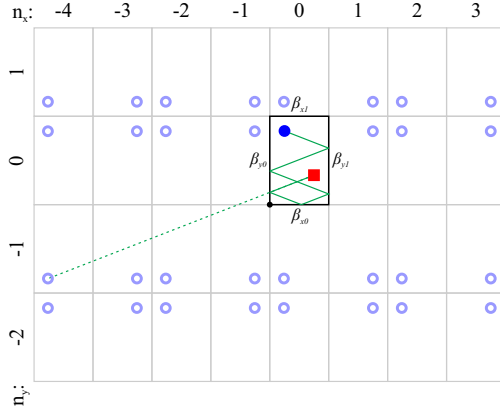

Fig. 1. Image sources for a two dimensional room. The red square and the blue dot represents the receiver and the source and the blue circumferences represents the image sources. The solid green line represents one of the multiple reflection paths and the dashed green line the direct path of the equivalent image source. The black dot is the origin of the coordinates system.

calculated using its grid indices; as an example, the component x would be calculated as:

$$x_{\mathbf{n}} = \begin{cases} n_x L_x + x_s & \text{if } n_x \text{ is even} \\ (n_x + 1)L_x - x_s & \text{if } n_x \text{ is odd} \end{cases}, \quad (1)$$

where $\mathbf{L} = (L_x, L_y, L_z)$ is the size of the room and $\mathbf{p}_s = (x_s, y_s, z_s)$ is the position of the original source. The the y and the z coordinates can be obtained equivalently.

The distance from the image source $\mathbf{n}$ to a receiver in the position $\mathbf{p}_r = (x_r, y_r, z_r)$, and therefore the delay of arrival, is trivial if we know the image source position:

$$d_{\mathbf{n}} = ||\mathbf{p}_r - \mathbf{p}_s||, \quad (2)$$

$$\tau_{\mathbf{n}} = \frac{d_{\mathbf{n}}}{c}, \quad (3)$$

where $|| \cdot ||$ denotes the Euclidean norm and c is the speed of sound.

In order to calculate the amplitude with which the signals from each image source arrive to the receiver, we need to take into account the reflection coefficients of the walls of the room. We define β_{x0} as the reflection coefficient of the wall parallel to the x axis closest to the origin of the coordinates system and β_{x1} as the farthest; β_{y0} β_{y1} β_{z0} and β_{z1} are defined equivalently. Finally, if we define $\beta_{\mathbf{n}}$ as the product of the reflection coefficients of each wall crossed by the path from the image source $\mathbf{n}$ to the receiver, the amplitude of the signal will be

$$A_{\mathbf{n}} = \frac{\beta_{\mathbf{n}}}{4\pi \cdot d_{\mathbf{n}}}. \quad (4)$$

Knowing the amplitude and the delay for each image, we can easily obtain the RIR as the sum of the contribution of each image source:

$$h(t) = \sum_{\mathbf{n} \in \mathcal{N}} A_{\mathbf{n}} \cdot \delta(t - \tau_{\mathbf{n}}), \quad (5)$$

where $\delta(t)$ is the Dirac impulse function.

B. Improvements to the original algorithm

1) Fractional delays: In order to implement (5), we need to deal with the fact that the values of $\tau_{\mathbf{n}}$ may not be multiples of the sampling period. The original algorithm proposed to just approximate the fractional delays by the closest sample, however, the error introduced by this approximation is too high for some applications, such as Sound Source Localization with microphone arrays. In [7], Pattersons proposed to substitute the Dirac impulse function by a sinc windowed by a Hanning function:

$$\delta'(t) = \begin{cases} \frac{1}{2}\left(1 + \cos\frac{2\pi t}{T_\omega}\right)\operatorname{sinc}(2\pi f_s t) & \text{if } -\frac{T_\omega}{2} < t < \frac{T_\omega}{2} \\ 0 & \text{otherwise} \end{cases}, \quad (6)$$

where f_s is the sampling frequency and T_ω is the window length. This is motivated by the low pass anti-aliasing filter that would be used if the RIR was recorded with a microphone in the real room. A window duration of $T_\omega = 4$ms is typically used.

Using the Patterson approach with $T_\omega = \infty$ is equivalent to compute (5) in the frequency domain as the sum of complex exponential functions as proposed in [8] [9], but using shorter window lengths reduces the computational complexity of the algorithm.

2) Negative reflection coefficients: Using positive reflection coefficients as proposed in [6] generates a low frequency artifact that must be removed using a high-pass filter. In addition, while a RIR recorded in a real room has both positive and negative peaks, all peaks generated by the ISM are positive. Using negative reflection coefficients as proposed in [9], solve both problems without need to add any posterior filter to the ISM algorithm.

3) Diffuse reverberation: To properly simulate a RIR, we need to use values of N_x, N_y and N_z high enough to get all the reflections in the desired reverberation time, but the number of operations in (5) grows in a cubic way. A popular solution is modeling the late part of the RIR, also know as diffuse reverberation, as a noise tail with the correct power envelope. In [10], Lehmann and Johansson propose using noise with logistic distribution and the formula introduced in [11] for the power envelop. However, for the sake of computational efficiency, we decided to use a simple exponential envelope following the popular Sabine formula [12].

III. PARALLEL IMPLEMENTATION

As shown in Fig.2, the parallel computation of the delays and amplitudes of arrival for the signals from each image source and their sinc functions is straightforward since there are not any dependencies between each source, and computing RIRs for different source or receiver positions in parallel is also trivial. However, the parallelization of (5) involves more problems, as the contributions of all image sources need to be added to the same RIR.

It is worth mentioning that, though it would be possible to compute in parallel RIRs from different rooms, we choose to implement only the parallelization of RIRs corresponding to

CUDA functions	Description	Time (%)
calcAmpTau_kernel	Equations (3) and (4)	0.82%
generateTime_kernel	Time vector computation	0.00%
generateRIR_kernel	Sincs computation and initial sum (5)	90.14%
reduceRIR_kernel	Parallel sum (5)	1.02%
envPred_kernel	Power envelope prediction	0.03%
generate_seed_pseudo	cuRAND function (diffuse reverberation)	7.88%
gen_sequenced	cuRAND function (diffuse reverberation)	0.01%
diffRev_kernel	Diffuse reverberation computation	0.03%
CUDA memcpy	[CPU to GPU]	0.00%
CUDA memcpy	[GPU to CPU]	0.06%

for the following 750ms. It can be seen how the bottleneck is located at the beginning of the computation of (5), which is due to the high amount of sinc functions that need to be computed. The following sections provide further details about the implementation of the different parts of the algorithm.

A. Amplitudes and delays computation

For computing (3) and (4), we use calcAmpTau_kernel, which computes sequentially each RIR but parallelizes the computation for each image source. Although parallelizing the computations for each RIR would have been possible, since $N_x \cdot N_y \cdot N_z$ is generally greater than the number of RIRs to compute, the level of parallelization is already quite high and, as shown in Table I, further optimizations of the kernel would have had a slight impact on the final performance of the simulation.

B. Computation and sum of the contribution of each image source

The computation of (5) is the more complex part of the implementation as it implies a reduction operation (the sum of the contributions of each image source), which is harder to parallelize, and a high number of nonlinear operations. We can see it as creating a tensor with 3 axis (each RIR, each image source, and each time sample) and summing it along the image sources axis. However, the size of this tensor would be huge and it would not fit in most of the GPUs memory.

To solve this problem, we first compute and sum a fraction of the source contributions sequentially, so the size of the tensor we need to allocate in the GPU memory is reduced; we do that through generateRIR_kernel. Specifically, this kernel performs the sum of 512 images in parallel for each time sample of each RIR.

After that, we use reduceRIR_kernel recursively to perform the reduction in parallel by pairwise summing the contribution of each group of images as shown in Fig.3. Performing the whole sum in parallel would lead to all the threads concurrently writing in the same memory positions, which would corrupt the result.

It can be seen in Table I how most of the simulation time is expended in generateRIR_kernel, this is due to the high amount of sinc functions that need to be computed, and also happens in the sequential implementations. However, thanks to the computing power of modern GPUs, we can compute many sinc functions in parallel and therefore reduce the time we would have needed to sequentially perform them in a CPU.

C. Diffuse reverberation computation

For the diffuse reverberation, we first use envPred_kernel to predict in parallel the amplitude and the time constant of each RIR. After that, we use the cuRAND library included in the CUDA Toolkit to generate a uniform distributed noise (the functions generate_seed_pseudo and gen_sequenced in Table I belong to this library) and we finally finally transform it to a logistic distributed noise and apply the power envelope through diffRev_kernel,
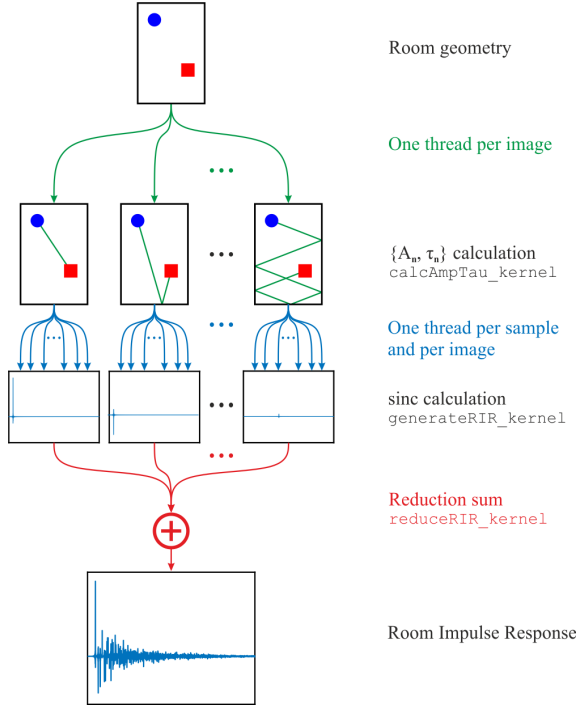

Fig. 2. ISM parallel implementation. Our library actually compute some of the sincs sequentially, which leads to a more efficient memory use. The reduction sum is detailed in Fig.3
.

the same room. This was because the number of image sources to be computed depends on the room dimensions and the reverberation time and to compute different rooms in parallel we would have need to use the worst case scenario (i.e. the smallest room and higher reverberation time) for all of them, which would have decreased the average performance.

In order to implement the ISM in GPUs, we decided to use CUDA [13]. Our code is divided in several CUDA kernels[2], Table I list them and the proportion of time employed in each one to compute 6 RIRs with $T_{60} = 1$s using the ISM method for the 250 first milliseconds and the diffuse model

[2]A CUDA kernel is a function that, when is called, is executed N times in parallel by N different CUDA threads in the GPU. For more details, see the CUDA programming guide: https://docs.nvidia.com/cuda/cuda-c-programming-guide/
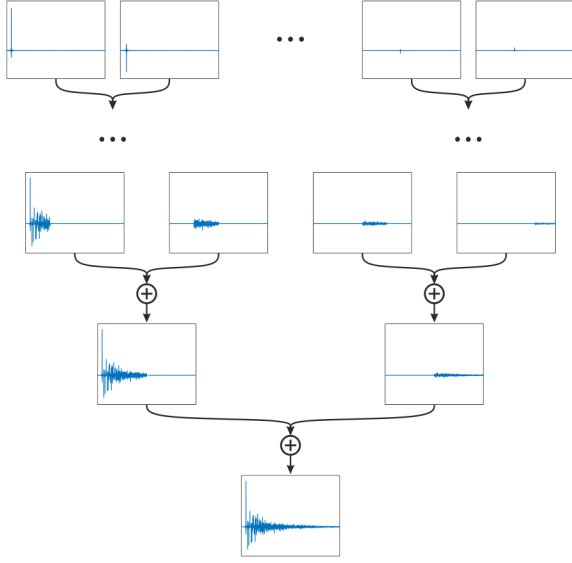
Fig. 3. Parallel reduction sum of the sincs (each level is performed by a call to `reduceRIR_kernel`). The sum must be performed pairwise to avoid several threads to concurrently write in the same variable. The sums of each time sample are also performed in parallel.

which parallelize the computations of each sample of each RIR.

D. Simulating moving sources

In order to simulate the recording of a moving source by a microphone array, we need to compute the RIR between each point of the trajectory and each microphone of the array and filter the sound source by them using the overlap add method. In sequential libraries, the complexity of the filtering is negligible compared to the RIR simulation; however, in our library, thanks to the performance of the GPUs, we found that we also needed to parallelize the filtering process if we did not want to be limited by its performance (specially for short reverberation times). To solve this problem, our library is able to compute multiple convolutions in parallel using the cuFFT library (included in the CUDA Toolkit) and a custom kernel to perform the pointwise multiplication of the FFTs.

IV. PYTHON LIBRARY

We have included the previous implementation in a Python library that can be easily compiled and installed using the packet manager and be used as any CPU library. The library provides a function which takes as parameters the room dimensions, the reflections coefficients of the walls, the position of the source and the receivers, the number of images to simulate for each dimension, the duration of the RIR in seconds, the time to switch from the ISM method to the diffuse reverberation model, and the sampling frequency and it returns a 3D tensor with the RIR for each pair of source and receiver positions. Information about the polar pattern of the receivers and their orientation can be also included in the simulation.

We also provide some python functions to predict the time when some level of attenuation will be reached, to get the

reflections coefficients needed to get the desired reverberation time (expressed in terms of T_{60}, i.e. the time needed to get an attenuation of 60dB), and to get the number of image sources to simulate in each dimension to get the desired simulation time without loss reflections. Finally, we include a function to filter a sound signal by several RIRs in order to simulate a moving source recorded by a microphone array.

V. RESULTS

In order to show the benefits of using GPUs for RIR simulation, we have compared our library against two of the most employed libraries for this purpose: the Python library presented in [14], whose code is freely available in [15], and has been used, for example, in [3] [16] [17], and the Matlab™ library presented in [10], whose code is freely available in [18], and has been used in [2] [19] [20] among others.

The Python library does not implement any kind of diffuse model, so it is expected to have worse performance than the MatLab™ library and our GPU library if we use it. The MatLab™ library uses the formula presented in [11] to model the power envelope of the diffuse reverberation, which is more complex than our exponential envelope model, so, for the sake of a fairer comparison, we modified the Matlab™ implementation to use a exponential model. The simulations with the sequential libraries and the ones with the Nvidia™ GTX 980Ti were performed in a computer with an Intel™ Core i7-6700 CPU and 16GB of RAM, while the simulations with the Nvidia™ Tesla P100 and V100 were performed in a n1-standard-2 instance in the Google Cloud Platform with 2 virtual CPUs cores and 7,5GB of RAM memory.

Fig.4 (a) represents the runtime of the different libraries for computing different number of RIRs in a room with size 3x4x2.5m and $T_{60} = 0.7$s. It can be seen how our library can simulate a hundred more RIRs in a second than the Matlab™ library even with a GPU designed for gaming (the Nvidia™ GTX 980Ti). Using our library without any kind of diffuse reverberation modeling we have a similar execution time than the Matlab™ library, which only computes the ISM until the RIR has an attenuation of 13dB, and we are also about a hundred times faster than the Python library.

In Fig.4 (b) we show the runtime of the different libraries for computing 128 RIRs in a room with size 3x4x2.5m and different reverberation times. We can see again how our library is about two orders of magnitude faster than the sequential alternatives. It is worth mentioning that our library has some limitations because calculating a large number of RIRs with high reverberation times may require more memory than it is available in the GPU; however, using the diffuse reverberation model, this limitation appears only for really high number of RIRs and reverberation times. Furthermore, it would be always possible to batch the RIRs in several function calls to circumvent this problem.

We did not perform any simulation campaign to analyze the acoustic accuracy of the library as it is the same of well known ISM methods, but we have compared our results against the cited libraries to validate it.
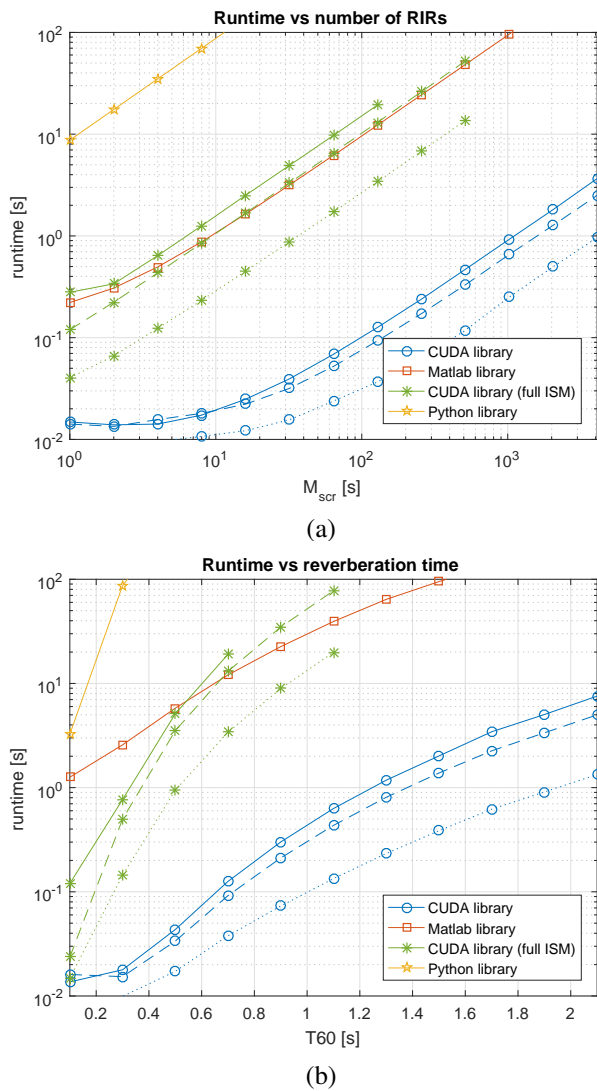
(a)

(b)

Fig. 4. Runtime of each library for computing (a) different numbers of RIRs in a room with size 3x4x2.5m and $T_{60} = 0.7$s and (b) 128 RIRs with different reverberation times. The solid lines were obtained using a Nvidia™ GTX 980Ti GPU, the dashed with a Nvidia™ Tesla P100 and the doted with a Nvidia™ Tesla V100.

VI. CONCLUSIONS

A new free and open-source library to simulate RIRs, which uses GPUs to dramatically reduce the execution time, has been presented. To the best of out knowledge, it is the first library with these features freely available on the Internet, and it could allow to the acoustic signal processing community, for example, to generate huge datasets of moving speaker speech signals in a reasonable computation time or to compute the acoustics of a Virtual Reality (VR) scene in real time.

REFERENCES

[1] C. Weng, D. Yu, S. Watanabe, and B. F. Juang, "Recurrent deep neural networks for robust speech recognition," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2014, pp. 5532–5536.

[2] Anthony Griffin, Anastasios Alexandridis, Despoina Pavlidi, Yiannis Mastorakis, and Athanasios Mouchtaris, "Localizing multiple audio sources in a wireless acoustic sensor network," *Signal Processing*, vol. 107, pp. 54–67, Feb. 2015.

[3] D. S. Williamson and D. Wang, "Time-Frequency Masking in the Complex Domain for Speech Dereverberation and Denoising," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 7, pp. 1492–1501, July 2017.

[4] Manfred R. Schroeder, "Natural Sounding Artificial Reverberation," *Journal of the Audio Engineering Society*, vol. 10, no. 3, pp. 219–223, July 1962.

[5] Zhong-hua Fu and Jian-wei Li, "GPU-based image method for room impulse response calculation," *Multimedia Tools and Applications*, vol. 75, no. 9, pp. 5205–5221, May 2016.

[6] J. B. Allen and D. A. Berkley, "Image method for efficiently simulating small-room acoustics," *The Journal of the Acoustical Society of America*, vol. 60, no. S1, pp. S9–S9, Nov. 1976.

[7] Patrick M. Peterson, "Simulating the response of multiple microphones to a single acoustic source in a reverberant room," *The Journal of the Acoustical Society of America*, vol. 80, no. 5, pp. 1527–1529, Nov. 1986.

[8] B. D. Radlovic, R. C. Williamson, and R. A. Kennedy, "Equalization in an acoustic reverberant environment: Robustness results," *IEEE Transactions on Speech and Audio Processing*, vol. 8, no. 3, pp. 311–319, May 2000.

[9] J Antonio, L Godinho, and A Tadeu, "Reverberation Times Obtained Using a Numerical Model Versus Those Given by Simplified Formulas and Measurements," *ACTA ACUSTICA UNITED WITH ACUSTICA*, vol. 88, pp. 10, 2002.

[10] E A Lehmann and A M Johansson, "Diffuse Reverberation Model for Efficient Image-Source Simulation of Room Impulse Responses," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 18, no. 6, pp. 1429–1439, Aug. 2010.

[11] Eric A. Lehmann and Anders M. Johansson, "Prediction of energy decay in room impulse responses simulated with an image-source model," *The Journal of the Acoustical Society of America*, vol. 124, no. 1, pp. 269–277, July 2008.

[12] Wallace Clement Sabine, *Collected Papers on Acoustics*, Cambridge : Harvard University Press, 1922.

[13] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron, "Scalable Parallel Programming with CUDA," in *ACM SIGGRAPH 2008 Classes*, New York, NY, USA, 2008, SIGGRAPH '08, pp. 16:1–16:14, ACM.

[14] Manuël A.P. Habets, "Room Impulse Response Generator," Tech. Rep., Sept. 2010.

[15] Marvin182, "Room Impulse Response Generator," https://github.com/Marvin182/rir-generator, Aug. 2018.

[16] A. Hassani, J. Plata-Chaves, M. H. Bahari, M. Moonen, and A. Bertrand, "Multi-Task Wireless Sensor Network for Joint Distributed Node-Specific Signal Enhancement, LCMV Beamforming and DOA Estimation," *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 3, pp. 518–533, Apr. 2017.

[17] S. Markovich, S. Gannot, and I. Cohen, "Multichannel Eigenspace Beamforming in a Reverberant Noisy Environment With Multiple Interfering Speech Signals," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 17, no. 6, pp. 1071–1086, Aug. 2009.

[18] Eric A. Lehmann, "Matlab implementation of fast image-source model for room acoustics," http://www.eric-lehmann.com/fast_ISM_code/, Oct. 2018.

[19] D. Pavlidi, M. Puigt, A. Griffin, and A. Mouchtaris, "Real-time multiple sound source localization using a circular microphone array based on single-source confidence measures," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2012, pp. 2625–2628.

[20] Anastasios Alexandridis, Anthony Griffin, and Athanasios Mouchtaris, "Capturing and Reproducing Spatial Audio Based on a Circular Microphone Array," *Journal of Electrical and Computer Engineering*, vol. 2013, pp. 1–16, 2013.