2D Room Acoustic Simulation with CUDA

Wei-Chih Hung[*] National Taiwan University

Wen-Hsiang Shaw[†] National Taiwan University

Yen-Cheng Chou[‡] National Taiwan University

Abstract

We implemented a 2D room acoustic simulation system, mainly referring the paper [Raghuvanshi et al. 2009]. The system is able to simulate the sound for a listener at a designated position in the given room map with the multiple sound sources. The 2D room map and the positions of the sound sources need to be designated in the process of pre-computed simulation. The system allows changing the position of the listener and the sound sources input in real time with the acceleration of CUDA. The propagation is simulated in frequency domain by exploiting the Acoustic Wave Equation numerically.

Keywords: acoustic simulation, sound propagation, computational acoustics, GPU

1 Introduction

To make virtual world sound more real, the ability to simulate room acoustic fast and accurately is important. As a result, computational acoustics has been an active research area for many years. To summarize the previous work, techniques for simulating acoustics can be classified into two parts: Geometric Acoustics and Numerical Acoustics.

Geometric Acoustics is based on the assumption of rectilinear propagation of sound waves, like light. All the geometric acoustics approaches suffer from the common drawback that it results in unphysical sharp shadows and fails to simulate diffraction.

On the other hand, Numerical Acoustics is based on Acoustic Wave Equation. Numerical Acoustics approaches are often computationally exhaustive but very suitable for parallel computation. In this project we use Numerical Acoustics approaches to implement 2D room acoustic simulation. The simulation firstly decomposes room space into several rectangles using Adaptive Rectangular Decomposition (ARD)[[**?**]] method. Each rectangle is transformed by Discrete Cosine Transform (DCT) to compute the sound propagation. The paper[[**?**]] showed that in this way, our simulation is faster and more accurate than FDTD method.

[*]e-mail: b96901036@ntu.edu.tw

[†]e-mail: b96901085@ntu.edu.tw

[‡]e-mail: b96901136@ntu.edu.tw

2 Algorithm

2.1 System overview

Our system overflow is shown in Fig.1.

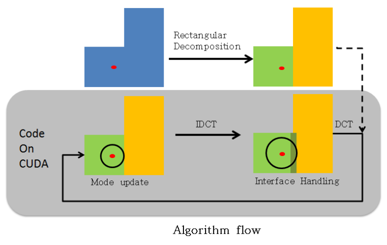

Figure 1: *System overview*

2.2 Rectangular decomposition

Traditional methods like FDTD accumulate numerical dispersion errors while calculating propagation on each cell. The dispersion errors can be eliminated by transforming the propagation from time domain onto frequency domain. Discrete Cosine Transform (DCT) is suitable for the transformation. However, DCT can only be performed on rectangles. As a result, a process called rectangular decomposition aims to decompose the room space into rectangles. After transformed onto frequency domain, the propagation introduces only the errors produced between two rectangles. The larger interface produces larger errors. To minimize the errors produced from the interface, the rectangular decomposition aims to minimize the interface length. A heuristic method is implemented that it randomly picks a cell called seed and then find the largest rectangle which contains only the unselected cell including the seed. Then the cells in the rectangle are labeled as selected. The above process is iterated until every cell is selected.

2.3 Propagation simulation

A loop is designed to simulate the propagation shown below.

1. Transform the pressure field and forcing term field onto frequency domain by DCT
2. Update the mode coefficients using the update rule
3. Transform the pressure field and forcing term field onto spatial domain by iDCT
4. Compute the forcing terms. For cells on the interface, use interface handling, and for the cells within point sources, use the sample value.

The loop is iterated for every time step. The four steps in the loop are illustrated in the following subsections.

2.3.1 DCT

After the rectangular decomposition, DCT is performed on the pressure and forcing term fields. The eigenfunctions of the DCT is given by

$$\Phi_i(x,y,z) = \cos\left(\frac{\pi i_x}{l_x}x\right)\cos\left(\frac{\pi i_y}{l_y}y\right)\cos\left(\frac{\pi i_z}{l_y}z\right) \quad (1)$$

$l_\xi = x,y,z$ stands for the sample step for each axis. The highest wavenumber eigenfunctions have to be spatially sampled at the Nyquist rate so that the discretization introduces no numerical errors for band-limited signals. The relation between two domains can be shown in the following equation,

$$p(\overrightarrow{x},t) = iDCT(M(\overrightarrow{x},t)) \quad (2)$$

$M(\overrightarrow{x},t))$ is the mode coefficients from the DCT.

2.3.2 Update rule

The propagation of sound is governed by the Acoustic Wave Equation,

$$\frac{\partial^2 p}{\partial t^2} - c^2 \bigtriangledown^2 p(\overrightarrow{x},t) = F(\overrightarrow{x},t) \quad (3)$$

$p(\overrightarrow{x},t)$ is the pressure field, c is the speed of sound which is $340ms^{-1}$ and $F(\overrightarrow{x},t)$ is the forcing term corresponding to sound sources. Eq. (1) can be reinterpreted in a spatial discrete setting as follows,

$$\frac{\partial^2 M_i}{\partial t^2} + c^2 k^2_{\overrightarrow{x}} M_i = F(\overrightarrow{x},t) \quad (4)$$

$$k^2_{\overrightarrow{x}} = \pi^2\left(\frac{i_x^2}{l_x^2} + \frac{i_y^2}{l_y^2} + \frac{i_z^2}{l_z^2}\right) \quad (5)$$

The sound sources can be modeled as a forcing term field $F(\overrightarrow{x},t)$. Assuming the $F(\overrightarrow{x},t)$ is constant over a time-step $\triangle t$, the forcing term field is transformed onto frequency domain shown in the following equation,

$$\overline{F_i}(t) = DCT(F(\overrightarrow{x},t)) \quad (6)$$

An update rule in a time discrete setting from Eq. (5) can be obtained by using the closed form solution of a simple harmonic oscillator over a time-step, shown in Eq. (7).

$$M_i^{n+1} = 2M_i^n\cos(\omega_i\Delta t) - M_i^{n-1} + \frac{2\overline{F_i^n}}{\omega_i^2}(1-\cos(\omega_i\Delta t)) \quad (7)$$

The errors might occur because of the assumption that the forcing term is constant over a time-step. However, if the time-step is below the sampling rate of the input signal, there is no error when updating the pressure and forcing term fields. There are only mode coefficients in three time step needed in Eq. (8) so in the system we only store the three latest mode coefficients. The pressure field at the current time step is stored separately.

2.3.3 Interface handling

After updating the pressure fields, the propagation between rectangles is simulated in a process called interface handling. Interface handling is based on Finite Difference approximation. We used sixth order accurate approximation shown in below,

$$F_i = \frac{c^2}{180h^2}(-2p_{i-2} + 27p_{i-1} - 270p_i + 270p_{i+1} - 27p_{i+2} + 2p_{i+3}) \quad (8)$$

i is the position on the axis normal to the interface and h is the grid spacing size. The forcing terms on the boundary in two adjacent rectangles are calculated by Eq. (8) to simulate the propagation between rectangles. Close look at Eq. (8) reveals that only two latest forcing terms fields need to be stored. The Finite Difference approximation introduces numerical errors.

2.4 Pre-computed simulation

Given a room's geometric map and the position of sound source(s), we can simulate the sound wave propagation of any input sound wave, but the simulation time can be really long even we use the above algorithm, which is much faster than other FDTD algorithm.

To solve this problem , we can compute the Impulse Response of every point in this room, denoted as $r_{ij}(t)$, where (i,j) is the position index of this room. If we want to compute the sound wave after sound propagation in this specific room map, $y_{ij}(t)$, where there is a given input sound, $x(t)$, at position (i,j), the following equation can be derived:

$$y_{ij}(t) = \int_{-\infty}^{\infty} x(\tau) r_{ij}(t-\tau)d\tau \quad (9)$$

That is, the output sound is simply the convolution of input sound the impulse response of the given receiver position.By this means, we can compute the output sound of any position while we had the pre-computed impulse response.

2.4.1 Gaussian simulator

Since this system can only simulate the limited frequency sound waves, use the discrete delta function as input to this simulation system will cause lots of distortion and thus the result cannot be treat as the impulse response of this system.

Thus, we use a band-limited Gaussian pulse as the input of the pre-computation, which is given by:

$$G(t) = exp(-\frac{(t-5\sigma)^2}{\sigma^2}) \quad (10)$$

where $\sigma = (\pi\eta_f)^{-1}$, and η_f is the upper-bond of the simulation frequency. In our case, it is set to 2000Hz.

2.4.2 Peak detection

Since our input signal is a low-pass impulse, the output impulse response will have no high-frequency term(fig.2).

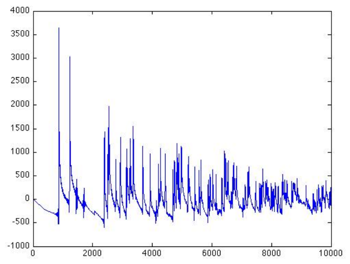

Figure 2: *Original impulse response of a Gaussian pulse input at a given receiver position*

Thus, we can use the peak-detection algorithm to reconstruct the high-frequency term (Fig.3). The peak-detection algorithm we used is attempting to find every local maximum peak value and if the peak value is higher than a time-varying threshold, it will be recognized as a peak and add to the final impulse response.

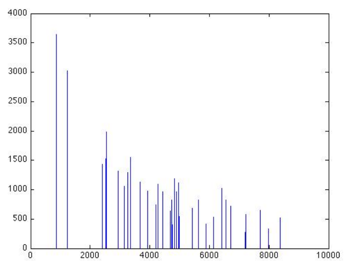

Figure 3: *Impulse response of fig.2 after peak detection*

2.5 Real-time demonstration

2.5.1 User interface

We created the user interface by QT (fig.4), this interface can let user choose the precompute maps, for now there are four candidates for the room maps. Users can click on the room map. If the click position is valid, the corresponding impulse response will be loaded to the program, while the background process keeps convolving the line-in sound samples and the impulse response and outputs to the line-out device.

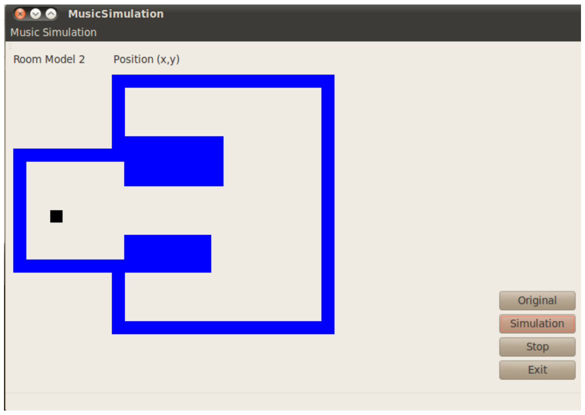

Figure 4: *User interface*

2.5.2 STFT and OLA method

For the real-time interactive system, we implement the Short Time Fourier Transform (STFT) and Overlap-Add Method (OLA) to achieve low latency in our playback system. For every time slot, there are N (Same as the length of the Impulse Response) samples retrieved from the Line-in device, at time t, denoted as $x_t[n]$. The pre-computed impulse response is denoted as $r[n]$, Then $x_t[n]$ and $r[n]$ are both padded zeros behind their tails:

$$x'_t[n] = \begin{cases} x_t[n], & 0 \leq n < N \\ 0, & N \leq n < 2N \end{cases} \tag{11}$$

and r'[n] is computed as the same from above equation. Padding zeros is necessary to the FFT-based convolution correctness. and y[n] is computed as:

$$y_t[n] = \text{IFFT}\{\text{FFT}\{x'_t[n]\} \cdot \{\text{FFT}\{r'[n]\}\}, \quad n = 0 \ldots 2N-1 \tag{12}$$

Then the output signal, denoted as $o_t[n]$, can be computed using OLA method:

$$o_t[n] = y_t[n] + y_{t-1}[n+N], \qquad n = 0 \ldots N \tag{13}$$

Finally, $o_t[n]$ is send to the Line-out device. As long as the computing time of the above algorithm is less than one time slot, this system can work in real time, and is suit for our demonstration.

2.5.3 Linux Sound API

Since our operating system is Linux, we chose the PulseAudio API, which is embedded to most part of popular Linux branches, to implement our record and playback system.

There are three main threads in our main program. First thread keeps catching the user input position and loading the corresponding impulse response. Second thread keeps retrieving audio samples from line-in device and pushing it to the process queue. The last thread keeps listening to the process queue, and performing the STFT and OLA method when the process queue is not empty.

3 CUDA acceleration

In this project, all the data is transformed between time-domain and frequency domain, which requires large computation on transformation. Moreover, transformation like DCT and FFT can be highly parallelized. In this project, we use *CUFFT* library on CUDA to accelerate our algorithm on DCT and Convolution.

3.1 FFT-based DCT with CUDA

DCT is an algorithm which can be highly parallelized. For the original equation, to transform a 2D $M \times N$ discrete plane to DCT domain, elements $F(u,v)$ on DCT domain can be derived as

$$F(u,v) = (\frac{2}{M})^{\frac{1}{2}}(\frac{2}{N})^{\frac{1}{2}} \sum_{i=0}^{M-1}\sum_{j=0}^{M-1} C_u(i)C(j)cos[\frac{\pi u}{2M}(2i+1)]cos[\frac{\pi u}{2N}(2j+1)] \tag{14}$$

To compute every element on the CPU, it would be $O(m^2n^2)$ algorithm. However, it can also be computed on GPU with time complexity $O(mn)$ if each element is computed within one thread simultaneously.

To reduce the computation time, *FFT-based DCT* is used instead of directly suming up over all the elements of DCT. We have expanded 1D DCT algorithm, whose time complexity is $O(m \log m)$, into 2D DCT algorithm based on the *CUFFT*. The total time complexity has decreased to $O(m \log m + n \log n)$

3.2 Convolution with CUDA

To make the convolution between impulse response and music input real-time possible, we transfer both impulse response and music data to GPU, use parallelized FFT algorithm [Kauker et al. 2010] to transform it onto frequency domain, multiplication element by element, then use IFFT to get the convolved response data.

4 Results and discussion

In our demonstration system, we have precomputed 4 room maps whose room sizes are about $20m \times 20m$. their spatial displacement h is set to about 0.1 meter, and the simulation frequency is set to

Algorithm 1: FFT-basedDCT

```
input  : An original data array of size $N$
output: An DCT data array of size $N$
1. Generate a sequence $y[m]$ from the given sequence x[m]:
```

$$y[m] = x[2m]$$
$$y[N-1-m] = x[2m+1](i = 0, \ldots, N/2-1) \quad (15)$$

```
2. Use CUFFT to transform y[m] into Y[n]
```

$$Y[n] = \mathcal{F}[y[m]] \quad (16)$$

```
3. Obtain X[n] from Y[n] by
```

$$X[n] = Re[e^{-jn\pi/2N}Y[n]] \quad (17)$$

Algorithm 2: 2D FFT-based DCT

```
input  : An original data matrix f of size $M \times N$
output: A DCT data matrix F of size $M \times N$
1. For $i \leftarrow 0$ to $M-1$ FFTbasedDCT($f_i$) ;
2. $f^T \leftarrow Transpose(f)$;
3. For $j \leftarrow 0$ to $N-1$ FFTbasedDCT($f_j^T$);
4. $F \leftarrow Transpose(f^T)$;
```

22050Hz which is half of 44.1kHz, Fig.5 is an snapshot of screen while we were precomputing the impulse response.

Since we have to store the impulse response for the whole room map, we decide to downsample the pressure field of every time slot, that is, every 16×16 grids will only be saved as one value after each time slot.

The duration of impulse response is 2 seconds, in our condition also 44100 samples, in order to preserve the whole reservation of the space. We have used microphone and music player to test our real-time demonstration system. The music after it was convolved with the impulse response sounds just like we were listening to it in a big empty room. The volume and reverebration will change while we move the receiver position through the user interface.

5 Conclusion

The major contribution of our work is that we accelerated the heavy computation of ARD, most DCT and IDCT, by computing them in parallel with CUDA on GPU. The computation avoids the transfer between GPU and CPU. As a result, our method reduces the computation time compared to the original ARD method.

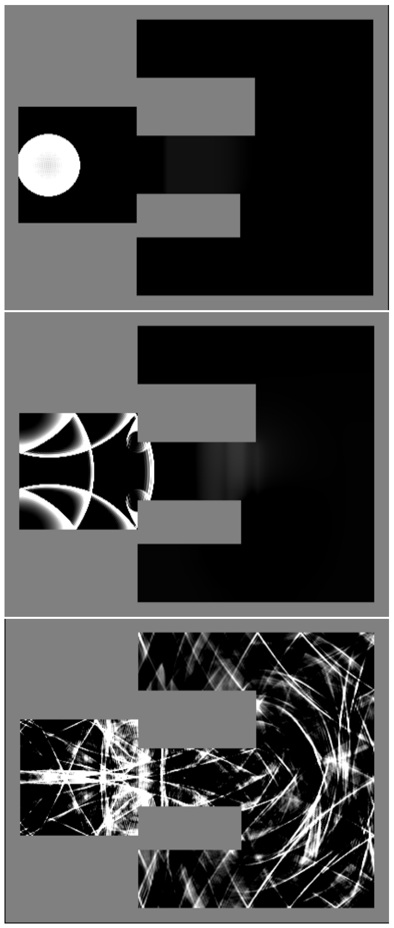

Figure 5: *the precomputing impulse wavefront*

References

KAUKER, D., SANFTMANN, H., FREY, S., AND ERTL, T. 2010. Memory saving discrete fourier transform on gpus. In *CIT*, 1152–1157.

RAGHUVANSHI, N., NARAIN, R., AND LIN, M. C. 2009. Efficient and accurate sound propagation using adaptive rectangular decomposition. *IEEE Trans. Vis. Comput. Graph. 15*, 5, 789–801.