Room Acoustic Simulation with CUDA

說明文件

Wei-Chih Hung 洪暐智
Wen-Hsiang Shaw 蕭文翔
Yen-Cheng Chou 周彥丞

Introduction

We implemented a 2D room acoustic simulation system. 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. With the acceleration of CUDA, the simulation process can be shorten a lot. 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.

Environment & Device

Computer : Laptop ASUS N82JQ
Operating System : Ubuntu 10.04LTS 32-bit
Cpu : Intel Core i7-720QM
Gpu : NVIDIA GEFORCE GT 335M

Content

project : The whole project source file and executables
project/gpu_src : Simulation with CUDA source file
project/cpu_src : Simulation in CPU source file
project/RoomModel : example simulation room models
project/DemoGui_src : real-time demo tool's gui source file
project/Demo_src : real-time demo tool's background source file
project/Demo : pre-build executables and simulation results(ubuntu 32bit)
project/bin : compile destination folders
docs/ : readme and introduction files

How To Compile

Request library :

CUDA SDK : in our test machine it is CUDA SDK 3.2 and CUFFT is needed
OPENCV : 2.2+ library is needed
QT : QT4 is needed for demo tools gui
pulse-audio : this library is usually pre-installed in current linux

Makefile configuration:

There are few library path(s) should be changed before compilation:
CUDA_SDK_PATH in project/gpu_src/Makefile
OPECV related PATH in every source folder's Makefile

Compile procedure:

```
>cd project
>clean_all.sh
>build_all.sh
```

after that, those binary files should located in bin/ folder and the Simulation_gpu and Simulation_cpu link files should auto link to the binarys.

How to Execute the Project

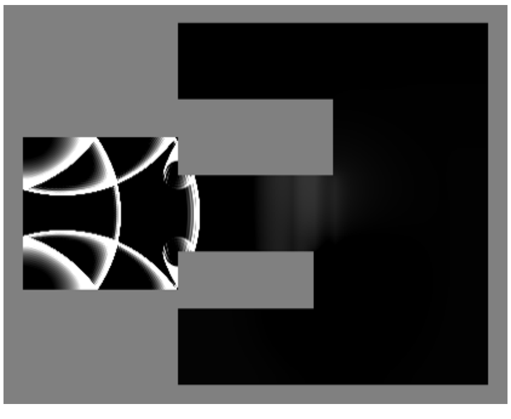
This section assumed that all the binary file have been build-up.

Simulation in GPU:

```
> cd project
> ./Simulation_gpu [room model file] [src_x] [src_y]    [output file]
```

where model file's example is placed in project/RoomModel, and src_x and src_y is the location of sound source. Output file is the raw impulse response output file, also the only output in this program

After execute the simulation, you should see the whole sound wave propagation in a window.



Simulation in CPU:

```
> cd project
> ./Simulation_gpu [room model file] [src_x] [src_y]    [output file]
```

After execute the simulation, you should see the same window as gpu simulation.

Demo tool:

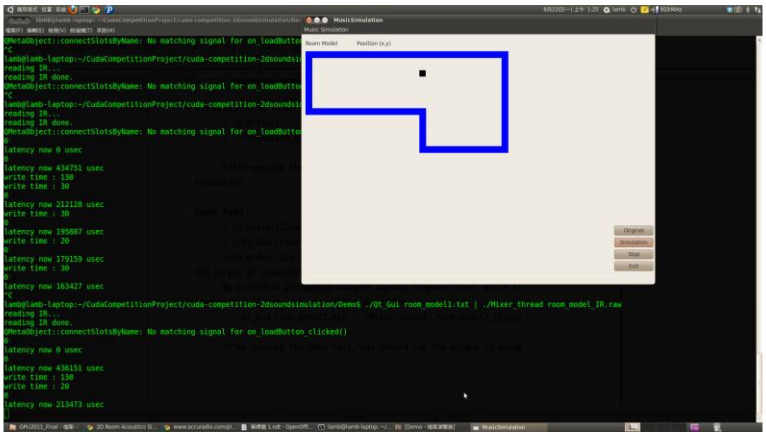
```
> cd project/Demo
> ./Qt_Gui [room model file] | ./Mixer_thread [impulse response file]
```

room model file is the room model text file, and impulse response file is the output of simulation file.

We provide a precompute example impulse response file, which is room_model2_IR.raw. One can simply use it by:

```
> ./Qt_Gui room_model2.txt | ./Mixer_thread    room_model2_IR.raw
```

After execute the Demo tool, you should see the window just like image in below:



```
QMetaObject::connectSlotsByName: No matching signal for on_loadButto
^C
lamb@lamb-laptop:~/CudaCompetitionProject/cuda-competition-2dsoundsi
reading IR...
reading IR done.
QMetaObject::connectSlotsByName: No matching signal for on_loadButto
^C
lamb@lamb-laptop:~/CudaCompetitionProject/cuda-competition-2dsoundsi
reading IR...
reading IR done.
QMetaObject::connectSlotsByName: No matching signal for on_loadButto
0
latency now 0 usec
0
latency now 434751 usec
write time : 138
write time : 30
0
latency now 212128 usec
write time : 30
0
latency now 195887 usec
write time : 20
0
latency now 179159 usec
write time : 30
0
latency now 163427 usec
^C
lamb@lamb-laptop:~/CudaCompetitionProject/cuda-competition-2dsoundsimulation/Demo$ ./Qt_Gui room_model1.txt | ./Mixer_thread room_model_IR.raw
reading IR...
reading IR done.
QMetaObject::connectSlotsByName: No matching signal for on_loadButton_clicked()
0
latency now 0 usec
0
latency now 436151 usec
write time : 138
write time : 20
0
latency now 213473 usec
```

One can change the receiver position by mouse click in the room (need a sort of drag), and this tool will capture the input sound of microphone and play it as you stand in the receiver position hear from the sound source position in this room model. There will be about 2 sec latency due to short time Fourier transform.

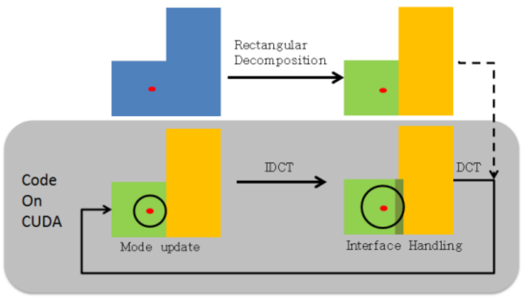
Algorithms introduction

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 approach 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 into several rectangles using Adaptive Rectangular Decomposition (ARD)[1] method. Each rectangle is transform by Discrete Cosine Transform (DCT) to compute the sound propagation. The paper[1] showed that in this way, our simulation is faster and more accurate than FDTD method.



Algorithm flow

Our system overflow is shown above.

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.

Propagation simulation

A loop is designed to simulate the propagation shown below.

(a) Transform the pressure field and forcing term field onto frequency domain by DCT
(b) Update the mode coefficients using the update rule
(c) Transform the pressure field and forcing term field onto spatial domain by iDCT
(d) 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.

DCT

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

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

$l_\xi, \xi = x,y,z$ 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,

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

$M(\bar{x},t)$ is the mode coefficient from the DCT.
iDCT is simply the inverse function of DCT which transformed the fields onto the spatial domain.

Update rule

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

$$\frac{\partial^2 p}{\partial t^2}-c^2\nabla^2 p(\bar{x},t)=F(\bar{x},t)\quad , \quad (4)$$

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

$$\frac{\partial^2 M_i}{\partial t^2}+c^2k_{\bar{x}}^2 M_i=F(\bar{x},t)\quad , \quad (5)$$

$$k_{\bar{x}}^2=\pi^2(\frac{i_x^2}{l_x^2}+\frac{i_y^2}{l_y^2}+\frac{i_z^2}{l_z^2})\quad , \quad (6)$$

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

$$\bar{F}_i(t)=DCT(F(\bar{x},t))\quad , \quad (7)$$

An update rule in a time discrete setting from Eq. (6) can be obtained by using the closed form solution of a simple harmonic oscillator over a time-step, shown in Eq. (8).

$$M_i^{n+1}=2M_i^n\cos(\omega_i\Delta t)-M_i^{n-1}+\frac{2\bar{F}_i^n}{\omega_i^2}(1-\cos(\omega_i\Delta t))\quad , \quad (8)$$

The errors might occur because of the assumption that the forcing term is constant over a time-step. However, if the time-step Δt 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.

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,

$$F_i = \frac{c^2}{180h^2}\left(-2p_{i-2} + 27p_{i-1} - 270p_i + 270p_{i+1} - 27p_{i+2} + 2p_{i+3}\right), \quad (9)$$

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. (9) to simulate the propagation between rectangles. Close look at Eq. (9) reveals that only two latest forcing terms fields need to be stored. the Finite Difference approximation introduces numerical errors.

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:

$$y_{ij}(t) = \int_{-\infty}^{\infty} x(\tau) r_{ij}(t-\tau)\, d\tau, \quad (10)$$

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.

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 this pre-computation, given by:

$$G(t) = \exp\left(-\frac{(t-5\sigma)^2}{\sigma^2}\right), \quad (11)$$

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

Peak detection

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

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.

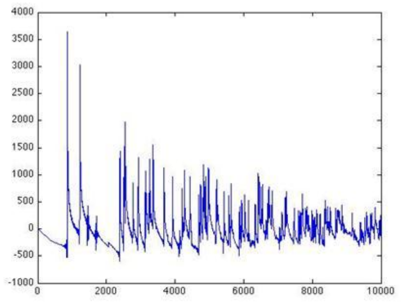


Figure 2: Original impulse response of a Gaussian pulse input at a given receiver position

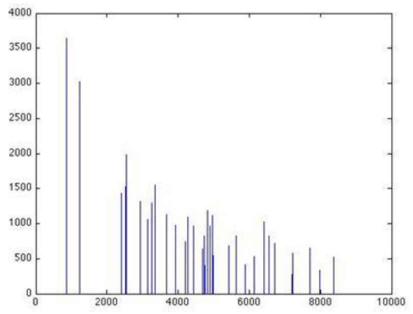


Figure 3: Original impulse response of a Gaussian pulse input at a given receiver position

User interface

We create the user interface by QT (Fig. 4), this interface can let user chose the pre-computed maps, for now there is four room maps can be selected. User can click on the room maps. If the click position is valid, the corresponding impulse response will be loaded to the program, while the background process keep convolving the line-in sound samples and the impulse response and output to the line-out device.

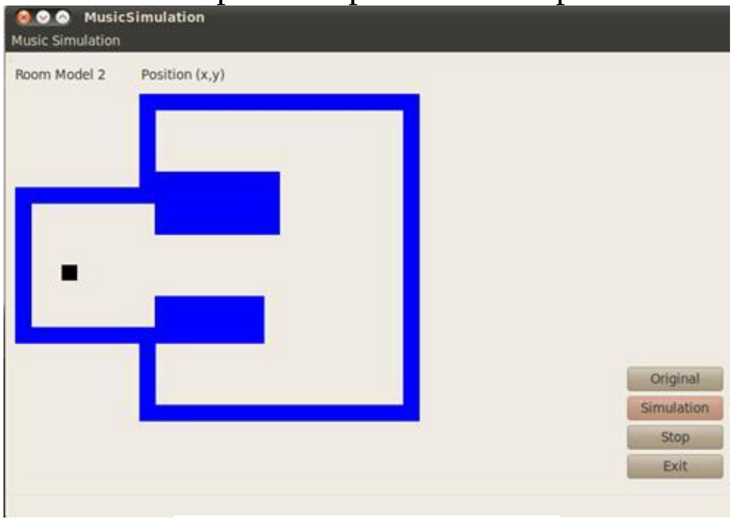


Figure 4: user interface by Qt

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 are 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:

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

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:

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

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

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

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.

Linux Sound API

Since our operating system is Linux, we choose 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 keep catching the user input position and load the corresponding impulse response. Second thread keeps retrieving audio samples from line-in device and pushes it to the process queue. The last thread keep listening to the process queue, and process the STFT and OLA method when the process queue is not empty.

CUDA acceleration

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

FFT-based DCT with CUDA

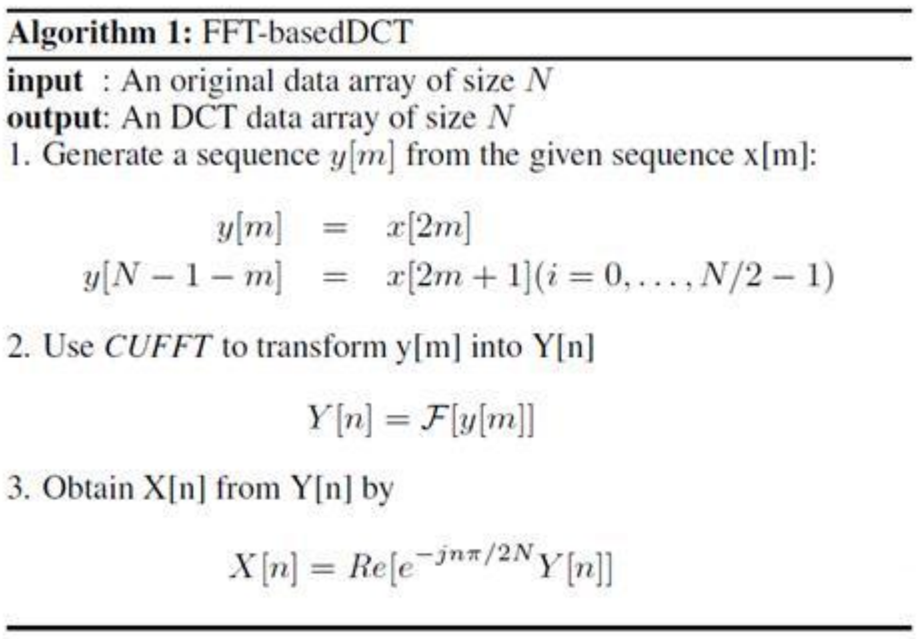
DCT is an algorithm which can be highly parallelized. For the original equation, to transform a 2D MxN discrete plan to DCT domain, element $F(u,v)$ on DCT domain can be derived as

$$F(u,v) = (\frac{2}{M})^{\frac{1}{2}}(\frac{2}{N})^{\frac{1}{2}}$$

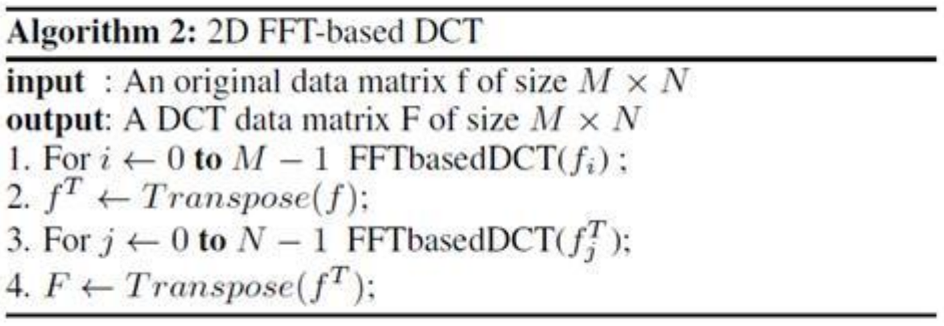
$$\sum_{i=0}^{M-1}\sum_{j=0}^{M-1} C_u(i)C(j)cos[\frac{\pi u}{2M}(2i+1)]cos[\frac{\pi u}{2N}(2j+1)]$$

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 with one thread simultaneously.

To reduce the computation time, *FFT-based DCT* is used instead of directly sum over all element of DCT. We have expand 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 $O(m\log m + n\log n)$



```
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]:
       $y[m] = x[2m]$
       $y[N-1-m] = x[2m+1](i = 0, \dots, N/2-1)$
2. Use $CUFFT$ to transform y[m] into Y[n]
       $Y[n] = \mathcal{F}[y[m]]$
3. Obtain X[n] from Y[n] by
       $X[n] = Re[e^{-jn\pi/2N}Y[n]]$
```



```
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. $F \leftarrow Transpose(f^T)$;
```

Convolution with CUDA

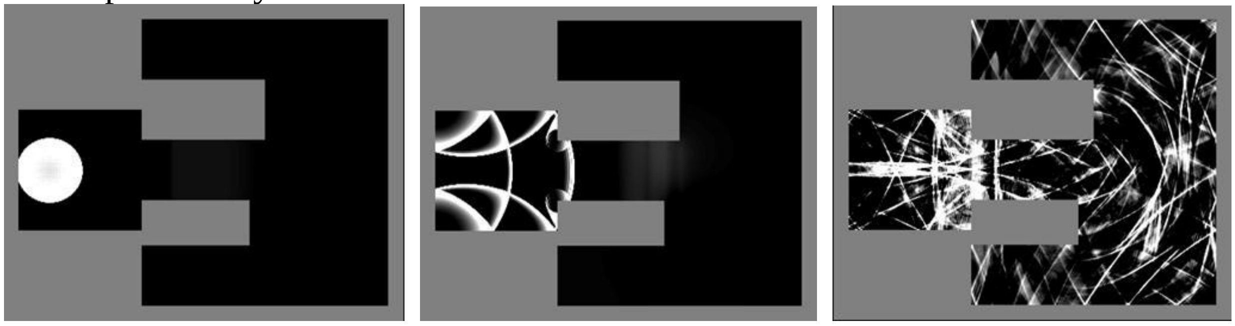
To make the convolution between impulse response and music input real-time possible, we transfer both impulse response and music data to GPU, use parallelized FFT algorithm to transform on to frequency domain, multiplication element by element, then use IFFT to get the convolved response data.

Results and discussion

In our demonstrate system, we have precomputed 4 room maps which their room size are all around 20m x 20m, their spatial displacement h is set to about 0.1 meter, and the simulation frequency is set to 22050Hz which is half of 44.1kHz, The figure below 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, so we decide to downsample the every time slot's pressure value matrix, that is , every 16 x 16 grids will only be saved one value after each time slot.

The impulse response length is 2 seconds, in our condition also 44100 samples, in order to preserve the whole reservation of the space. We had use microphone, music player to test our real-time demonstration system, the music after convolved with the impulse response sound just like we were listening to it in a big empty-room, and the sound volume and reverberation will change while we are moving the receiver position by the user interface.



Conclusion

The major contribution of our work is that we accelerate 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.

Reference

[1]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.
[2]KAUKER, D., SANFTMANN, H., FREY, S., AND ERTL, T. 2010. Memory saving discrete fourier transform on gpus. In *CIT*, 1152–1157.

Experiment Results

There are two models in this experiment. We compared the simulation time between original cpu simulation and CUDA accelerated gpu simulation. It turns out there will be a near 2 times speed up!

Room models	**CPU time (sec)**	**CUDA time(sec)**	**Acceleration**
Room 1	483.81	184.38	2.62x
Room 2	477	266.98	1.78x

This experiment is done with the previous said ASUS N82JQ, with a mobile GPU core. We think the CUDA version will be even faster if we use the Desktop Computer’s GPU.