Physically-based Sound Synthesis on GPUs

Qiong Zhang[1], Lu Ye[1, 2] and Zhigeng Pan[1]

[1]College of Computer Science, Zhejiang University, Hangzhou 310027, China
[2]Department of Computer and Electronic Engineering, Zhejiang University of Science and Technology, Hangzhou 310012, China
E-mail: zgpan@cad.zju.edn.cn

Abstract. Modal synthesis is a physically-motivated sound modeling method. It has been successfully used in many applications. However, if large number of modes are involved in a simulated scene, it becomes an overwhelming task to synthesize sounds in real time without special hardware support. An implementation based on commodity graphics hardware is proposed as an alternative solution by using the parallelism and programmability in graphics pipeline.

1 Introduction

Sound has long been acknowledged as an effective channel in human-computer interaction [1-4]. Among all the sound synthesis approaches available today, physically based methods play an important role and have pretty long history in computer music research [5]. However, due to the computational complexity, it only became research target until very recently in interactive applications (e.g. games) [6] [7].

Doel et al [6] developed a system for automatic generating sounds made by contact interactions between solid objects. This system is based on a good physically-motivated model called modal synthesis. It models a vibrating object by a bank of damped harmonic oscillators that are excited by an external stimulus. O'Brien et al [7] extended this method by automatic calculating model parameters through finite element method.

Though modal synthesis can be performed efficiently with an O(N) algorithm for objects with N modes, real-time synthesis is only feasible if an interactive application has very small number of sounding objects [8]. Special hardware is necessary in order to simulate complex audio scenes.

Although GPUs (Graphics Processing Units) are specifically designed for transforming, rasterizing and texturing geometry primitives, they are becoming a popular platform for general-purpose computation due to inherent parallelism and enhanced programmability. Audio and signal processing is among one of the latest applications of GPUs [9]. Audio Video Exchange (AVEX) from BionicFX (http://www.bionicfx.com/) converts digital audio into graphics data, and then performs efficient calculations using the 3D architecture of GPUs. Jedrzejewski and

Marasek [9] implemented ray tracing on the GPU to accelerate computation of room impulse response that can later be used for auralization.

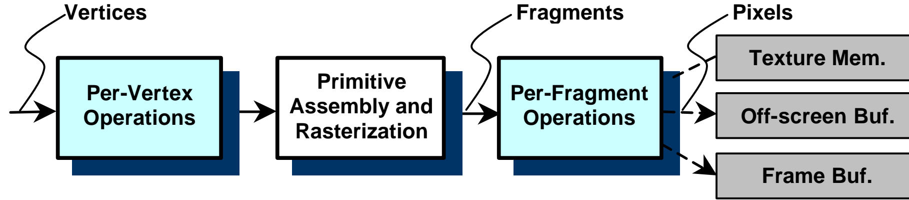

Fig. 1. Graphic pipeline on modern GPUs

As illustrated in Fig 1, a modern graphics pipeline is responsible for transforming input geometric primitives into a final image. The image then can be rendered to the frame buffer or stored in an off-screen buffer. Among the major steps of the whole process, we are particularly interested in the fragment-processing stage. At this stage, we can plug in a piece of code called fragment program specifically designed for each individual application. It can perform floating-point vector arithmetic on all the pixels in a parallel mode while having direct access to texture memory.

This paper proposes a GPU-based implementation for modal synthesis. Considering that each mode can be synthesized independently of others, it makes modal synthesis a very appealing target on GPUs. The basic idea is to pre-calculate modal models and store them in a 2D texture, and then fragment programs are used to do the actual response calculation.

2 GPU-based Modal Synthesis

2.1 Modal model representation on GPUs

Contact sounds produced by a solid object depend on quite a few of factors. Those factors can be roughly classified into two groups, namely static and dynamic group. The static group is independent of interaction, which includes geometry and material properties of an object. By contrast, the dynamic group includes factors that rely on current interaction and simulation context. Contact location and external force to produce sound on an object are two primary examples in this group.

As we know, an object in modal model can be represented with following set of parameters $\{ \boldsymbol{w}_i ,\ d_i ,\ A_i^j \}$ [6], where $1 \le i \le N$ represents the number of modes, and $1 \le j \le K$ represents sampling contact locations on the object. $\boldsymbol{w}_i$ is the mode frequency, d_i is the decay rate, and A_i^j is the mode gain at each particular location under an impulse force. All those parameters can either be derived manually through measurement [6] or automatic calculation based on a handful of geometry and materials properties of an object [7]. The number of modes and sampling locations are usually dependent on simulation context and resource availability. The corresponding

impulse response at each sampling location j is $y_j(t)=\sum_{i=1}^{N} A_i^j e^{-d_i t}\sin(\boldsymbol{w}_i t)$. Fig 2 shows an object and its first four modes.

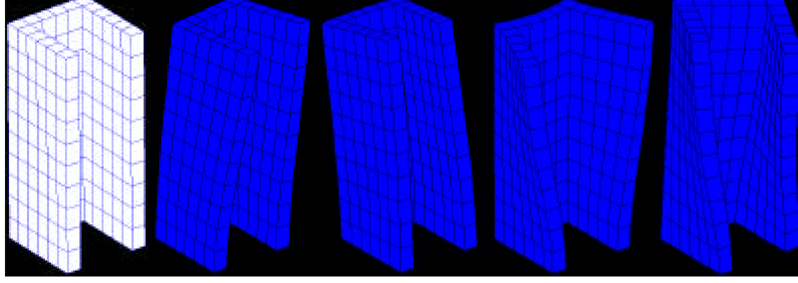

Fig. 2. An object in original form (the leftmost figure) and its first four modes

If we assume that the whole model is linear, the response under an arbitrary force $F(t)$ can be derived fully on the sampling impulse responses and represented in following discrete and recursive form:

$$y(t)=\sum_{i=1}^{N} 2COS_i y_i(t-1)-(COS_i^2+SIN_i^2)y_i(t-2)+SIN_i A_i^j F(t-1)$$

Where $SIN_i = e^{-d_i/S}\sin(\boldsymbol{w}_i/S)$, $COS_i = e^{-d_i/S}\cos(\boldsymbol{w}_i/S)$, and S is the audio sampling rate. It can be further represented with following formula:

$$y(t)=\sum_{i=1}^{N} y_i(t)=\sum_{i=1}^{N} R_i y_i(t-1)+G_i y_i(t-2)+B_i^j F(t-1)$$

Where $R_i = 2e^{-d_i/S}\cos(\boldsymbol{w}_i/S)$, $G_i = -(e^{-d_i/S})^2$, $B_i^j = e^{-d_i/S}\sin(\boldsymbol{w}_i/S)A_i^j$. All the R_i and G_i values can be pre-calculated before any actual simulation. The only simulation related factor is A_i^j in B_i^j which is dependent on contact location. The solution we take is to pre-calculate B_i^j at all the sampling locations. By this way, the calculation of B_i^j during any simulation becomes choosing a pre-calculated value correspondent to the sampling location closest to the actual contact location.

After all the sounding objects are represented in the form of modal models, we can store those model parameters in a 2D texture on GPUs. Basically, texture is a 2D array and each individual element of the texture called texel that may have up to four (RGBA) channels. As long as the maximum number of modes among all the objects is less than the maximum allowable texture width, multiple modal models can be stored in a single 2D texture. As illustrated in Fig 3, a three-channel texel is used to stored parameters (R_i, G_i, B_i^j) for each mode. Each row contains all the modes parameters of a sounding object at a specific contact location. Typically, 10-30 sampling contact locations are enough for common objects. Thus, a maximum-size 2D texture on modern GPUs can hold at least dozens or even hundreds modal models.

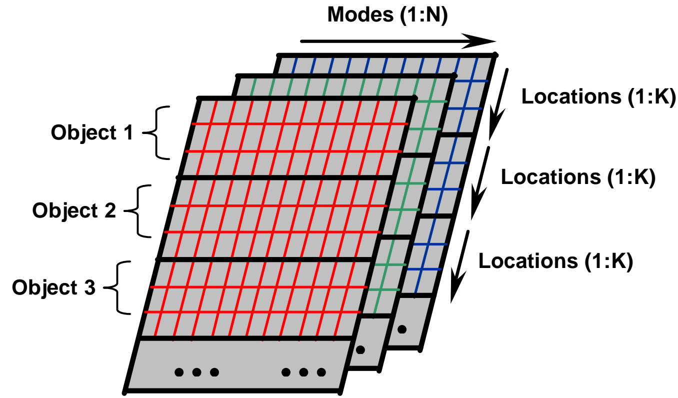

Fig. 3. Modal models are stored in a 2D texture

As we can see, this representation leads to some data redundancy since each R_i and G_i are stored in multiple rows as long as the number of sampling location is greater than 1. The purpose of this is to avoid texture addressing and simplify computational logic in the fragment program used to calculate response.

Before any actual simulation begins, all the modal models involved only need be transferred to a GPU once. All the values are in 32-bit floating-point format.

2.2 Modal synthesis on GPUs

It is a two-step process to perform modal synthesis on GPUs. The first step is to calculate the response for each individual mode from all the objects producing sound. The second step is to summarize all the responses from those objects.

The first step is accomplished by rendering a rectangle through a fragment program. Essentially, each fragment in the rectangle is corresponding to a mode and rendering a fragment is actually calculating current response for the mode. Each row in the rectangle contains the responses from all the modes of a single object. The total row number is the same with the number of objects producing sound in a scene. By this way, all the responses can be calculated independently in a parallel way.

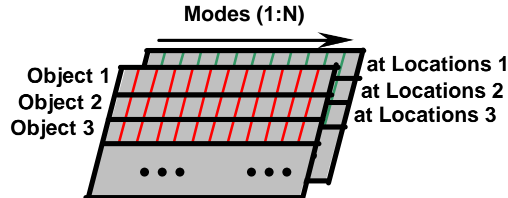

Fig. 4. Responses from all the modes are stored in a 2D texture

As illustrated in Fig 4, the responses from all the modes of all the objects are stored as a 2D texture in an off-screen buffer through render to texture (RTT) functionality. Each texel can be used to store up to four consecutive responses for a single mode. However, only two most recent responses, i.e. $y_i(t-1)$ and $y_i(t-2)$, are necessary to calculate current response $y_i(t)$.

```
CalcResponse(MTex,RTex,L,F)
1  A=MTex(L_row,col)   /*(Ri, Gi, Bi)*/
2  B=RTex(row,col)+F_row as the third component /*(yi(t-1), yi(t-2), F)*/
3  yi(t)=dot(A, B)
4  RTex(row,col)=(yi(t), yi(t-1),1)
```

Code List 1 – calculate response for each mode

Assuming that M is the number of objects producing sound, `MTex` is the texture storing all the modal models and `RTex(row,col)` is the response texture coordinates/fragment window position, Code List 1 gives pseudo code to calculate response for each mode.

During interactive simulation, there are two dynamic parameters in array type passed into the fragment program. One is location index $L_{1...M}$ (i.e. row number in `MTex`). It is used to address correct row/texel in `Mtex` that corresponds to the current contact location. The other is the magnitude $F_{1...M}$ of contact forces. Each object may have its own contact force. The response calculation for each mode is actually a dot product between two vectors, i.e. (R_i , G_i , B_i^j) and ($y_i(t-1)$, $y_i(t-2)$, $F(t-1)$). It is considered as a very efficient operation on GPUs. The calculation result $y_i(t)$ together with $y_i(t-1)$ is written back to the same texel, which will be used in the next cycle.

The second step is to get the total response by accumulating the responses from all the modes. It is essentially a reduction operation. Due to the fact that there is no global register or hardware accumulator on GPUs, reduction operation is commonly implemented as a multi-pass ping-pong process [10]. Starting with the initial rectangle, in each following step a rectangle scaled by a factor of 0.5 is rendered.

More specifically, in the fragment program, a sum value is evaluated from each four adjacent texels and written into a new texture, which is now of a factor of two smaller in each dimension than the previous one. For a texture in $N*N$ resolution, $\log_2 N$ rendering passes are performed until the final sum is obtained in a single pixel. It can then be read back to the system. In our scenario, since the number of modes is usually much larger than the number of objects, we may apply different factors in each dimension to reduce pass number during the ping-pong process.

3 Results and Conclusion

We have implemented above algorithm in NVIDIA Cg and a GeForce 6800 GT graphics processor with 256 MB video memory was used in the preliminary test. All the modal models were transferred to the card through an AGP 8X interface. Our test

cases include 64 sounding objects in the scene. Each object has 16 sampling contact locations and 512 modes. It is equivalent to synthesize 32K modes. Fig 5 gives a synthesis example that includes 10 sounding objects. However, on a typical workstation with Pentium IV 2.8G HZ CPU, we are only able to synthesize less than 5000 modes with similar algorithm in real time.

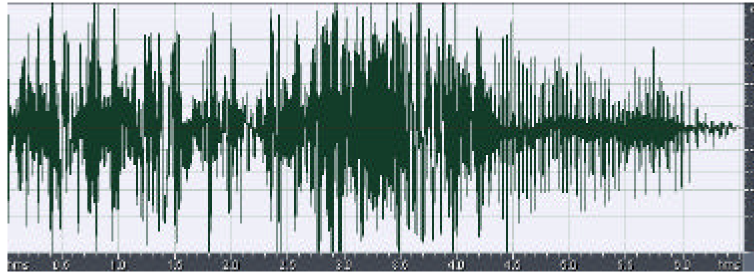

Fig. 5. A modal synthesis example on Geforce 6800 GT

If large chunk of audio data need be transferred from GPUs to the system memory, slow AGP read-back may become a performance bottleneck. However, newly appeared PCI-Express should be able to solve this issue.

Overall, the implementation of modal synthesis on GPUs is able to significantly improve the number of modes that can be synthesized in real time. With the constant increase of programmability (global register, more floating-point support, etc), fragment pipelines, and data transfer bandwidth between GPU and system memory, it is expected that the GPU based implementation can get further performance gain compared with the CPU one. Future research may focus on integrating modal synthesis and 3D filtering on GPUs.

Acknowledgments

This project is co-supported by a 973 project (grant no: 2002CB312100) and Excellent Youth Teacher Program of MOE in China.

References

1. Buxton, W.: Introduction to this special issue on nonspeech audio. Human Computer Interaction. 4 (1999) 1–9
2. Pan, Z., Shi, J.: Virtual Reality Technology Development in China: An Overview. International Journal of Virtual Reality. 4(2000) 2-10
3. Zhang, Q., Shi, J., Pan Z.: ARE: An audio reality engine in virtual environment. International Journal of Virtual Reality. 4 (2000) 37-43
4. Zhang, Q., Shi, J.: Progressive Sound Rendering in Multimedia Applications. Proceedings of 2004 IEEE International Conference on Multimedia and Expo (ICME 2004), v1, p651-654, Taiwan (2004)
5. Smith, J.: A Physical Modeling Synthesis Update. Computer Music Journal. 20 (1996) 44-56

6. van den Doel, K., Kry, P. Pai, D.: FoleyAutomatic: Physically-based Sound Effects for Interactive Simulation and Animation. Proc of SIGGRAPH 2001, 537-544, Los Angeles, USA (2001)
7. J. O'Brien, C. Shen, and C. M. Gatchalian, Synthesizing Sounds from Rigid-Body Simulations. Proc of 2002 ACM SIGGRAPH Symposium on Computer Animation, 175-182, San Antonio, USA (2002)
8. van den Doel, K., Knott, D., Pai, D.: Interactive Simulation of Complex Audio-Visual Scenes. Presence. 13 (2004) 99-111
9. Jedrzejewski, M., Marasek, K.: Computation of Room Acoustics Using Programmable Video Hardware. International Conference on Computer Vision and Graphics 2004, Warsaw, Poland (2004)
10. Krüger, J., Westermann, R.: Linear Algebra Operators for GPU Implementation of Numerical Algorithms. ACM Trans. Graph. 22 (2003) 908-916