Use of General Purpose Graphics Processors for Realtime Sound Synthesis

Marc Sosnick, Department of Computer Science
Faculty Advisor: William Hsu, Department of Computer Science

San Francisco State University

INTRODUCTION

- Powerful Graphics Processing Units (GPUs) are now common in standard graphics cards of most desktop and laptop systems.
- Many of these graphics cards support general-purpose computing using these powerful GPUs.
- Software support (CUDA [6], OpenGL [7]) now exists to utilize GPUs for general-purpose computing.
- We have been exploring the use of GPUs for realtime sound synthesis using compute-intensive physics-based models, previously impossible in realtime using CPUs.
- Others have used physics-based models generate audio [2,3], *but none have executed in realtime*.
- Realtime sound synthesis using compute-intensive physics-based models will allow the creation of new audio synthesizer instruments.

SYNTHESIS METHOD

To generate sound, we simulate the propagation of waves in 2-dimensions using the wave equation with dissipation:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \frac{\partial^2 u}{\partial t^2} + \eta \frac{\partial u}{\partial t} \qquad (1)$$

We approximate this using the truncated second-order Taylor expansion [1]:

$$\frac{u^n_{i+1,j} - 2u^n_{i,j} + u^n_{i-1,j}}{\Delta l^2} + \frac{u^n_{i,j+1} - 2u^n_{i,j} + u^n_{i,j-1}}{\Delta l^2} = \frac{u^{n+1}_{i,j} - 2u^n_{i,j} + u^{n-1}_{i,j}}{\Delta t^2} + \eta \frac{u^{n+1}_{i,j} - u^{n-1}_{i,j}}{2\Delta t} \qquad (2)$$

Since the grid is symmetric, $\Delta l = \Delta x = \Delta y$, and $x = i\,\Delta x$, $y = j\,\Delta y$, and $t = n\,\Delta t$ [5]. Solving for $u^{n+1}_{i,j}$:

$$u^{n+1}_{i,j} = \left[1 + \tfrac{\eta \Delta t}{2}\right]^{-1} \left\{ \begin{array}{c} \rho\left[u^n_{i+1,j} + u^n_{i-1,j} + u^n_{i,j+1} + u^n_{i,j-1} - 4u^n_{i,j}\right] \\ +2u^n_{i,j} - \left[1 + \tfrac{\eta \Delta t}{2}\right] u^{n-1}_{i,j} \end{array} \right\} \qquad (3)$$

A sample point is chosen on this simulated membrane, much like a needle point on a record. The sample point's displacement is sent to the audio buffer, which combine to create a sound wave (Figure 1).

The equation (3) is repeated as many times as necessary by the GPU to fill the buffer. This buffer full of samples is then returned to the CPU, which sends the buffer to the audio driver, which plays the sound. To avoid jitter (Figure 3), the time spent filling the buffer must be less than the time to play the buffer.

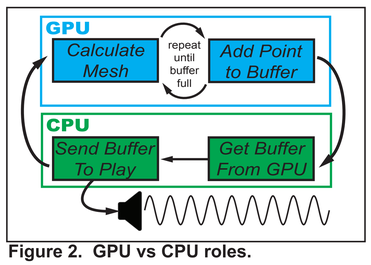

Figure 2. GPU vs CPU roles.

The membrane is put into motion with an initial offset (Figure 1, time = 0). This is roughly analogous to striking a drum head, or plucking a string.

EXPERIMENTAL SETUP

These experiments represent the first stage of our research. This is proof-of-concept work intended to show the feasibility of this approach.

We tested our software on three different systems (Table 1), equipped with mid-range graphics cards with GPU computing capability.

System	Graphics Card	CPU @ Clock Rate	GPU Cores @ Clock Rate	Pinned Memory	System Memory
1	GTX285	Intel Core 2 Quad @ 2.5 GHz	240 cores @ 1.48 GHz	✓	
2	9400M	Intel Core Duo @ 1.86 GHz	16 cores @ 0.80 GHz	✓	✓
3	8800GT	Intel Quad Xeon @ 3 GHz	112 cores @ 1.5 GHz		

Table 1. System configurations tested

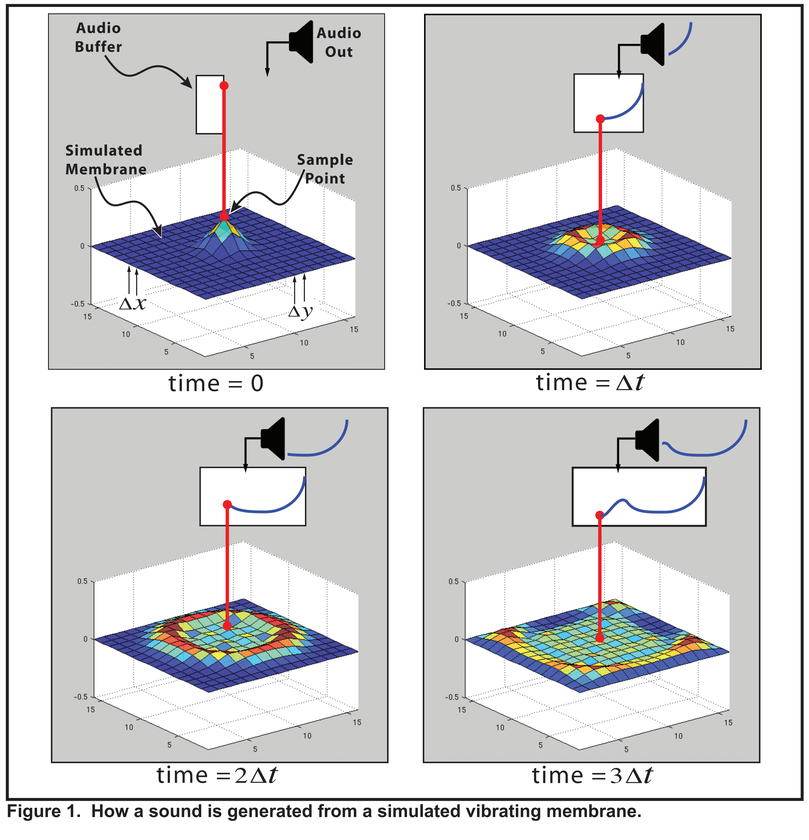

Figure 1. How a sound is generated from a simulated vibrating membrane.

We timed execution on the CPU and GPU with a variety of buffer sizes and grid sizes. We timed for latency and checked for jitter, which together define responsiveness [4].

Jitter: gaps or skips in the sound, usually due to the buffer not being filled sufficiently fast. To be responsive, zero jitter is acceptable. [4]

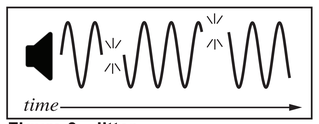

Figure 3. Jitter

Latency: the delay in starting or stopping a sound after a stimulus such as a key press. To be responsive, a delay of 20 - 30 ms is acceptable [4].

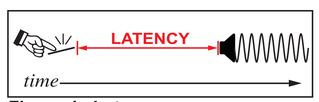

Figure 4. Latency

Jitter and latency (responsiveness) must be within acceptable limits to be able to use the instrument in realtime.

EXPERIMENTAL RESULTS

System	Processor	Configuration: Buffer (samples)	Configuration: Grid (points)
1 (GTX285)	CPU	≥ 4096	*
		*	≥ 20 x 20
	GPU	∅	∅
2 (9400M)	CPU	= 4096	*
		≤ 1024	*
	GPU	≤ 1024	*
3 (8800GT)	CPU	≤ 1024	= 21 x 21
	GPU	≤ 1024	= 21 x 21

Table 2. Buffer and grid configurations which produced jitter (* denotes all configurations).

System	Buffer Size (Samples)	GPU Time (ms)	Memory Transfer (ms)	GPU Total (ms)	CPU Time (ms)
GTX285	8	1626	0	1626	3060
	512	1062	0	1062	3032
	4096	1067	0	1067	3102
9400M	8	7251	0	7251	4052
	512	5674	0	5674	4088
	4096	2842	0	2842	4133
8800GT	8	2863	705	3568	2562
	512	2095	12	2106	2518
	4096	2110	2	2112	2539

Table 3. Constant 21 x 21 point grid with varying buffer size.

System	Grid Size (Points)	GPU Time (ms)	Memory Transfer (ms)	GPU Total (ms)	CPU Time (ms)
GTX285	15 x 15	924	0	924	1577
	18 x 18	984	0	984	2224
	21 x 21	1067	0	1067	3102
9400M	15 x 15	2222	0	2222	1984
	18 x 18	2957	0	2957	3040
	21 x 21	2842	0	2842	4133
8800GT	15 x 15	1411	2	1413	1266
	18 x 18	1743	3	1746	1843
	21 x 21	2110	2	2112	2539

Table 4. Constant 4096-sample buffer with varying grid sizes.

N.B. Time measurements in Tables 3 and 4 are cumulative measurements of a 1-second sample played 5 times.

CONCLUSIONS

- It is possible to generate realtime audio using GPUs and finite-difference simulations.
- Larger grids better leverage GPU computing power.
- Choice of buffer and grid sizes is important to responsiveness.
- Memory bandwidth is not a major consideration, especially with more advanced graphics cards.
- It should be possible to create a responsive, realtime synthesizer instrument using compute-intensive physics-based models.

FUTURE WORK

- Develop and optimize parallel algorithm to process arbitrarily large or dense grids.
- Write code in OpenCL to leverage heterogeneous computing environments and embrace industry standards.
- Package code into a modular, production-quality synthesis package.

REFERENCES

[1] A. Adib: "Study Notes on Numerical Solutions of the Wave Equation with the Finite Difference Method," *arXiv:physics/0009068v2 [physics.comp-ph]. 4 October 2000. Downloaded from http://arxiv.org/abs/physics/0009068v2 on April 15, 2010.*

[2] S. Bilbao: "A finite difference scheme for plate synthesis," *Proceedings of the International Computer Music Conference*, pp. 119-122, 2005.

[3] B. Land: "Finite difference drum/chime," Downloaded 4/15/2010 from http://instruct1.cit.cornell.edu/courses/ece576/LABS/f2009/lab4.html.

[4] N. P. Lago, F. Kon: "The Quest for Low Latency," *Proceedings of the International Computer Music Conference*, pp. 33-36, 2004.

[5] E. Motuk, R. Woods, S. Bilbao, J. McAllister: "Design Methodology for Real-Time FPGA-Based Sound Synthesis," *IEEE Transactions on Signal Processing*, Vol. 55, No. 12, pp. 5833 – 5845, 2007.

[6] *Nvidia CUDA Programming Guide, version 2.3.1. 8/26/2009.* Downloaded 4/21/2010 from http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/Nvidia_CUDA_Programming_Guide_2.3.pdf.

[7] *Nvidia OpenCL Programming Guide, version 2.3. 8/27/2009.* Downloaded from http://www.nvidia.com/content/cudazone/download/OpenCL/Nvidia_OpenCL_ProgrammingGuide.pdf

ACKNOWLEDGEMENTS

This research would not have been possible without the generous support of William Hsu, Niki Jorgensen, Blair Whitmer, Dragutin Petkovic, Russ Altman, Kai Kolhoff, Mike Wong, Carlo Matar, and Marcia and Myron Sosnick.