ICA 2013 Montreal
Montreal, Canada
2 - 7 June 2013

Underwater Acoustics
Session 5aUW: Using Graphic Processing Units for Computationally Intensive Applications in Acoustic Modeling and Signal Processing

5aUW10. Reverberation modeling on graphics processing units (GPUs)
Paul Hursky* and Ahmad T. Abawi

*Corresponding author's address: Heat, Light, and Sound Research Inc, 3366 North Torrey Pines Ct., La Jolla, CA 92037, paul.hursky@hlsresearch.com

We will discuss modeling reverberation on GPUs. An accompanying talk will discuss using GPUs to model scattering from rough surfaces. Here we discuss using GPUs to model the two sets of propagation paths, one set from the source to scatterer, the other set from the scatterer to the receiver, and to combine these two sets via a scattering operation and assemble the reverberation waveform at the receiver. We will discuss how we have adapted various aspects of our modeling to a GPU platform. GPUs provide several differentiated memory architectures that an application can exploit. For example, the texture memory provides hardware-assisted interpolation - as a result, we can load a 3D environment into texture memory, sparsely sampled, and then reconstruct interpolated slices as needed the modeling task. GPU architectures have been evolving for many years to meet the demands of computer gaming and rendering, applications most ambitiously served by ray tracing (of light). As a result, NVIDIA provides a ray tracing framework called OptiX, sufficiently general-purpose, that it can be linked with externally provided functions to specialize the mathematics implemented during the ray trace process. We will describe our work on adapting OptiX to underwater acoustic ray tracing.

Published by the Acoustical Society of America through the American Institute of Physics
INTRODUCTION AND BACKGROUND

General Purpose Graphical Processing Unit (GPGPU) programming is rapidly being adopted as an effective means of accelerating applications that lend themselves to parallel implementation. A decade ago, this strategy required strenuous programming (see Ref. 1) to organize numerical algorithms like FFTs at a low level on hardware specialized for rendering graphics. As the emphasis changed from designing hardware for specific graphics effects for use in computer gaming to more generic designs that would support a broader blend of effects, including as yet undiscovered applications, software frameworks like NVIDIA’s Compute Universal Device Architecture (CUDA), Portland Group’s Fortran CUDA, and the Khronos consortium’s OpenCL were developed to provide a higher-level programming interface that was less specialized for graphics and more amenable for general purpose numerics. Today, many scientific applications are being implemented using CUDA, CUDA Fortran and OpenCL as a result of increasing availability of libraries providing fundamental building blocks like LAPACK/BLAS and FFTs. As this progress was being made in GPGPU applications, computer gaming and professional graphics rendering have increasingly incorporated more realistic physics, with the major graphics hardware manufacturers providing software frameworks to make this more accessible. An example of such a software framework is NVIDIA’s OptiX ray tracing framework, used for realistic rendering, but which can be used to implement scientific ray tracing, and which can interoperate with CUDA’s GPGPU software within the same runtime context. We will report on our work combining CUDA and OptiX to accelerate reverberation modeling. This work was motivated by a need to synthesize multi-static reverberation fields suitable for operator training, where the transmit and receive platforms are in motion.

APPROACH

The architecture for modeling reverberation consists of:

- calculating the propagation from the transmitter to every potential scatterer in the space, which typically included the ocean surface, the seabed, and some distribution of volume scatterers,
- calculating the propagation from the receiver to every potential scatterer in the space,
- combining transmit-to-scatterer and scatterer-to-receiver (assuming reciprocity) paths through a scattering interaction that calculates the level of scattered sound as a function of arriving vertical angle, departing vertical angle, and, optionally, the azimuth of the outgoing path relative to the incoming plane of propagation
- combining all propagating components (rays or normal modes) at the receiver with the appropriate combined travel times, phase shifts and levels for each combination of arriving and departing propagating component (see Ref. 2)

The propagation modeling must thus reproduce angles relative to the scatterer, as well as transmission losses and phase shifts. Rays (see Ref. 3) and normal modes (see Ref. 4) have been used for modeling reverberation, since both representations can be used to produce these parameters. Implicit in this approach is that only a single out-of-plane scattering interaction contributes from each combination of transmit-to-scatterer and scatterer-to-receiver paths/modes, although specular reflections are used along each of the two paths.

This work leverages the advances in Graphical Processing Unit (GPU) technology and the software frameworks provided by CUDA and OptiX to accelerate both the propagation modeling and the final combination of contributions to form the reverberant waveform at the receiver.

EXPLOITING SPECIFIC FEATURES OF GPU HARDWARE

Figure 1 shows the actual physical hardware architecture in GPUs. However, the number of physical processing cores is typically much smaller than the actual number of threads that are executing in parallel. Figure 2 shows how the software threads are organized, in blocks of threads, and in grids of blocks, where each thread knows where it is within the two-level hierarchy of blocks and grids. GPU programs consist of a host program that downloads data to the GPU, kicks off the array of software threads, all running a SIMD program called a “kernel”, and then retrieves the processing results. Each software thread performs a small subset of the total workload, with each thread knowing which particular subset to work on, based on where it is located in the multi-dimensional array of blocks and grids.
FIGURE 1. Diagram of GPU hardware, showing multiple multi-processor units, each with a fixed number of cores, with fast shared and register memory, and scheduled to execute “virtual” threads of execution by a thread manager.

FIGURE 2. Diagram of GPU programming model, showing “virtual” threads organized into thread blocks (1D, 2D, or 3D), with the thread blocks themselves organized into “grids” (1D or 2D). The work is divided by mapping the data itself to “virtual” threads, which execute a SIMD program, or “kernel”, in which each “virtual” thread knows what data it must process, based on its location within the array of blocks and grids.

As in most hardware, GPUs have a hierarchy of memories, from very fast, limited-in-size, on-chip register memory to very slow, large size, off-chip global memory. In addition, there are specialized memories, such as texture memory and constant memory. Texture memory is useful for our modeling purposes, because it provides various hardware-assisted interpolation and filtering operations on its contents. For example, parameters of a 3D ocean can be loaded into texture memory, including a 3D sound speed field and bathymetry. This information is typically provided at a much coarser grid spacing than is typically needed in the modeling calculations. Fortunately, the texture memory can retrieve interpolated values for arbitrary locations.

Inevitably, in the course of its execution, a software thread must access data from one of the slower memories, resulting in that thread “blocking” until the memory access completes. To cope with such high latency events, and not have the underlying physical cores sit idle, the GPU thread scheduler immediately swaps another thread that is
ready to continue its execution. Unlike the threads on CPUs, the software threads on GPUs are extremely lightweight, so that swapping them on and off physical cores is extremely efficient, especially compared to threads on CPUs. It is important to program a GPU application so that there is a large number of software threads all vying for a limited number of physical cores, so that when a thread encounters a high latency wait, another thread is immediately swapped onto its physical core, so that the physical cores are kept busy most of the time. However, it is important that all the threads are running the same kernel. One consequence of this is that conditional events (e.g. programmed as if-then-else statements) result in divergent execution – since the threads must execute the same kernel, threads with one logical result execute the if block, while the threads with the other logical result sit idle, until the if-block code completes, and then execute the then-block code while the first set of cores sit idle. 

Although ray tracing may initially seem to be easily made parallel, it turns out that the work load among rays is very unequal, depending on what part of the ocean they encounter. As a result, it is a challenge to parallelize ray tracing on GPUs. Some rays are quickly “extinguished” due to being absorbed into the seabed. Others propagate for long distances. Fortunately, this is true for the mainstream commercial applications of ray tracing (modeling light propagation for realistic rendering of 3D objects), and as a result, a variety of “acceleration structures” have been developed to streamline the work. NVIDIA’s OptiX software framework is the result of many years of research whose goal is real-time ray tracing to render realistic 3D scenes using GPUs. OptiX provides a software framework for ray tracing, including a number of advanced “acceleration structures”, such as kd-trees and bounding volume hierarchies. We are investigating how to integrate ray tracing and Gaussian beam modeling used in underwater acoustics into the OptiX framework. This entails building a display-list representation of the ocean, including the surface, the sound speed layers, and the seabed, so that it can be incorporated into OptiX. OptiX provides accelerated ray tracing, but allows for user-provided software to be incorporated into OptiX, to be executed when various events are detected. For example, when a ray is detected to have intersected a sound speed layer, our code to calculate how such a layer refracts the ray is executed, and the various ray parameters that are germane to propagation through the ocean are updated in the user-specified ray payload. As this is work in progress, it remains to be seen whether the additional programming effort results in a significant acceleration of ray tracing in aid of reverberation modeling.

REFERENCES

8. NVIDIA CUDA C - Programming Guide. NVIDIA, v5.0, May 2012.