Fast and Accurate Time-Domain Simulations with Commodity Graphics Hardware

Gerard S. Baron*1, Costas D. Sarris1, and Eugene Fiume2

1 The Edward S. Rogers Sr. Department of Electrical and Computer Engineering,
2 Department of Computer Science, University of Toronto. Toronto, ON, M5S 3G4,
Canada

We show the efficacy of graphics hardware for the computational electrodynamics community. In particular, we illustrate how this hardware can be used to accelerate a general purpose Finite-Difference Time-Domain (FDTD) simulation through a set of image processing operations. While achieving a significant speedup, our simulation yields comparable accuracy to that of a consonant software implementation.

Introduction

Recent developments in graphics hardware acceleration provide operations that can be applied to dramatically speed up FDTD simulation. Although specialized hardware has been previously proposed for FDTD [1], the economy of scale gained by 3D graphics hardware due to the burgeoning video-game/entertainment industry provides a considerable cost/performance advantage. Graphics hardware is very fast, very cheap, and continues to be enriched at rates that outstrip those of general-purpose CPUs and FPGAs.

While Graphics Processing Units (GPUs) are designed for a specialized task, applying them to general-purpose computing has attracted wide research interest [2]. Recently, GPUs were introduced in the field of FDTD simulation as a tool for implementing 2D update equations for an isotropic medium [3]. Our work extends the approach of [3] by fully utilizing a state-of-the-art GPU in order to model anisotropic dispersive media, such as the Uniaxial Perfectly Matched Layer (UPML) absorber. Concepts from image processing are presented that intuitively relate graphics acceleration to FDTD simulation. Then the performance of a GPU accelerated simulator is evaluated in terms of both speed-up and accuracy. Thus, the well-known challenges in working with GPUs, namely an unusual programming model that caters to video game development [4] and proprietary architectures with largely unspecified bounds on error, are comprehensively addressed.

FDTD as Image Processing

The key to using GPUs for FDTD acceleration is interpreting FDTD update equations as spatial filters or kernel convolutions. In general, kernel convolutions are operators on arrays of previous states, and ‘spatial’ refers to how their input arrays are indexed. We shall see that image processing operations, which routinely makes use of 2D spatial kernel convolutions, can completely realize FDTD simulations. This is done by carefully defining images that encapsulate field components and material properties into colour channels. Afterwards, update equations are simply kernels that access and update these images. Furthermore, the feedback required to model time-marching can be achieved by procedurally reapplying kernels.

GPU Architecture

Developed for real-time 3D graphics, modern GPUs try to perform in hardware the computationally extensive operations involved in visualizing a scene from 3D data.
The speed of these devices has in the past taken precedence over their accuracy; many shortcuts are taken that distinguish real-time from realistic 3D graphics. One very powerful shortcut is texture mapping. Texture mapping is a technique that involves draping an image over a 3D model as opposed to providing more geometric detail. Modern GPUs allows textures to be dynamically generated with programmable image processing kernels called Pixel-Shaders. Two other noteworthy GPU features are floating-point image formats and support for high-precision feedback [5].

2D-UPML implementation

We implement a general purpose 2D-FDTD simulator, supporting UPML absorbing boundary conditions, satisfying the \( \text{TE}_y \) case based on Taflove et al’s formulation [6]. The simulation region is a rectangle on the xz-plane bounded by PEC walls, and excited by transparent point sources. The material properties over the region, its dimensions, the position of the point sources, and \( \Delta x \), \( \Delta z \), and \( \Delta t \) are simulation parameters. We present a high-level design that can be applied to more general simulations (i.e. 3D), and implemented on any GPU providing the features outlined in the previous section. Mesh size is only limited by GPU video memory. We coded in OpenGL with vendor specific extensions, and used NVIDIA’s high-level shading language, Cg.

The design involves three steps. First, we decide how to map the spatial dimensions of the simulation region to the dimensions of the image files processed by GPUs. For our 2D simulation region we map the width of the image to the x-direction and the height of the image to the z-direction, and define the lower-left corner of the image as the origin. A 3D simulation might employ a sequenced array of 2D images to reflect the extra dimension. The second step involves appropriately packing field values and material properties into images files and color channels. A summary of our five image files and their formats is illustrated in Fig.1(a). Design goals include minimizing the memory footprint of images, and grouping values in a way that minimizes the number of inputs used by the Pixel-Shaders. Finally the third step is designing Pixel-Shaders that procedurally update Efield and Hfield. We implemented four shaders: UpdateE, SourceE, UpdateH, and VisualizeE. An overview of the entire process is illustrated in Fig.1(b).

Simulation Results

The accuracy and speed of our accelerated simulator was benchmarked against a reference software implementation. We evaluated square-meshes, \( \Delta x = \Delta z = 0.0025 \text{ m} \) and stability limit of 0.9, with PEC walls and 16-cell UPML absorbers. A 0-20 GHz point source was placed at the centre of the mesh as illustrated in Fig.2(a). Our accelerated simulator supported single-precision floating-point. In contrast our reference simulator was coded at both a higher double-precision, and quasi single-precision (storage only) to provide base-line and consonant results respectively. The tests were run on an Intel P4 2.4GHz CPU with 512MB RAM and NVIDIA’s GeForce 6800GT GPU with 256MB video memory.

Fig.2(b) plots the accuracy of accelerated and consonant simulations as a function of time-step. Accuracy was measured by calculating the Euclidean Norm of the
component-wise error in the $E_y$-field mesh against base-line simulation results. Error was calculated for a 128x128 mesh every 8 time-steps over an 8192 time-step period. We attribute the common oscillatory characteristic of both curves to the periodic attenuation of reflected waves on the UPML absorbers. On average the accelerated simulator accumulated error 12.9434 times faster than the consonant simulator. This divergence is consistent with the fact that both reference simulators performed all arithmetic in double precision, and our speculation that the GPU was limited entirely to single-precision in hardware.

Fig.2(c) plots the simulation time in seconds of accelerated and consonant simulators for different meshes as a function of time-step. The accelerated simulator approaches an average sustained 17.5337 times speed-up as simulation size increases.

While results are still preliminary, these results provide lower bounds on accuracy and speed. For instance our accelerated simulator was developed to be instructive. As a result, it is less arithmetically precise or equivalently uses more floating-point operations, than our reference simulators. Besides eliminating these extra operations, there remain numerous optimizations to apply for improving accuracy and speed. If accuracy is of greatest concern, higher levels of precisions can be achieved by judiciously breaking up operations into multiple lower precision steps. While this would sacrifice some speed, we still anticipate an overall performance gain. In terms of performance, real gain will be achieved when complex data analysis is also adapted for the GPU. GPUs excels at pipelining batched operations with limited feedback. For example, performance results including a post-process visualization would have unfairly overwhelmed our reference implementations, but are virtually free for the accelerated simulator. Finally, current trends suggest GPUs will continue to improve in speed and precision.

**Conclusion**

We have shown that GPUs can be used to accurately accelerate a general-purpose FDTD simulation. Moreover, we have introduced a conceptual framework relating GPU acceleration in terms of image processing operations to FDTD simulation.
Besides experimenting with optimizations, further studies on accuracy, and adapting common data analysis methods for the GPU, future work includes applying this framework to higher-order methods like Multi-Resolution Time-Domain (MRTD).

References


