Acceleration of Asymptotic Computational Electromagnetics Physical Optics - Shooting and Bouncing Ray (PO-SBR) using CUDA™

Huan-Ting Meng[1], Jianming Jin[1], and Eric Dunn[2]

(1) CCEML, Dept. of ECE, University of Illinois at Urbana-Champaign

(2) Science Applications International Corporation

Introduction

Many defense and commercial communication system applications rely on electromagnetic simulations for signal exploitation and mission planning. For instance, when deploying a communication system it is important to predict the coverage that the system will provide. Such prediction often takes the form of field profiles, which are color plots indicating the simulated field strength from a transmitter over a geographic area.

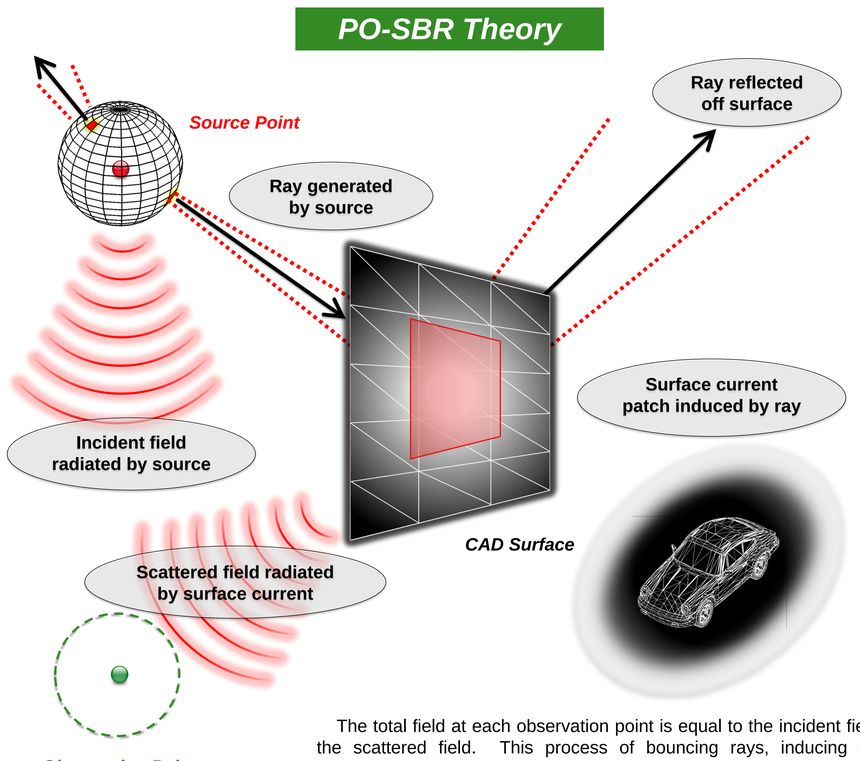
Despite their importance, field profiles can be difficult to generate. Physical measurements take time and often are impossible if access to the area is restricted. For numerical simulation based on computer models, even the fastest algorithms often take significant time to carry out the computation. This is because field values must be found not only for a large number of observation points, but also for multiple frequencies.

With the right choice of simulation algorithm, the field value at each observation point/frequency pair can be computed independently of all other observation point/frequency pairs, allowing parallel processing to be applied. One such algorithm is what is known as the **Physical Optics – Shooting and Bouncing Ray** (PO-SBR) algorithm. This algorithm involves two separate processes which themselves can be further accelerated with parallel processing:

1. SBR:
   - CAD models are used to represent the environment in which the fields will be measured.
   - Rays are launched spherically from each transmitter and traced as they bounce off and hit new CAD surfaces.
2. PO:
   - Intersection points where rays have bounced off surfaces get replaced with an equivalent current.
   - Equivalent currents then are radiated to each field observation point as a function of the signal frequency.

To apply parallel processing to this problem, a Quadro® FX 5800 device was provided by NVIDIA®. Two different versions are implemented, one using CUDA™ with a standard CPU ray tracer that takes advantage of the GPU shared memory, and another that uses only the GPU global memory but with NVIDIA®'s OptiX™ package for its ray tracer. Both versions were able to use this GPU's 240 cores (4GB global memory, 30 multiprocessors) to simulate field profiles over **150x** faster than with a standard CPU.

PO-SBR Theory



The total field at each observation point is equal to the incident field plus the scattered field. This process of bouncing rays, inducing surface currents, and radiating fields is repeated for all frequencies, all rays, and all observation points.

CUDA™ Implementation

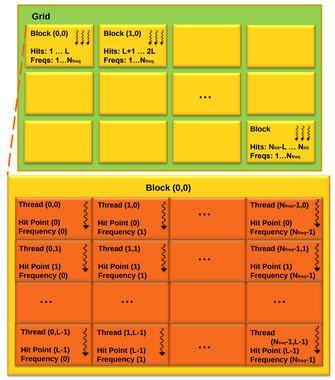
1. Tracing of rays

Millions of rays are launched from the source and are allowed to bounce around the scene which can be described by millions of triangular CAD facets.

- Version 1: Implemented on CPU with modified version of PBRT code (www.pbrt.org). Traces all rays to completion and passes entire ray history (N_{hit} hit points, surface normals, and ray directions) to GPU. The rest of the CUDA™ implementation shown here is for this version.
- Version 2: Implemented on GPU with NVIDIA®'s OptiX™ package (www.nvidia.com/object/optix.html). Traces a subset of all rays and retains ray history in global memory of GPU. This version was presented at HPEC workshop (www.ll.mit.edu/HPEC/2010).

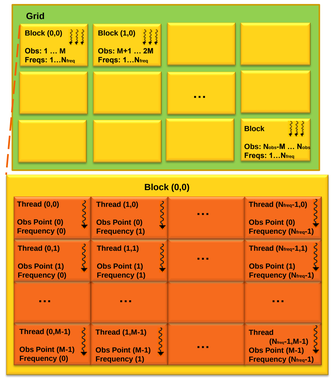
2. Radiation from source to all N_{hit} ray hit points

The incident field from the source needs to be computed at every initial ray hit point, and then its phase updated for all subsequent ray bounces in order to find the surface currents.



- A 2D grid of blocks is allocated (2D is to avoid grid dimensional limit).
- Each block in the grid computes the **incident field** at **L** **ray hit points** for all N_{freq} frequencies.
- The exact number of ray hit points per block is based on the amount of shared memory available per block.
- For a single frequency simulation this is around **110 hits per block**. More frequencies require more memory and therefore allow fewer ray hits per block.
- Each thread within the block (maximum of 512) computes the incident field for one combination of frequency and hit point.
- The field values for all frequencies and all hit points are stored in device global memory to be used by the next step.

3. Radiation from all N_{hit} ray hit points to all N_{obs} observation points

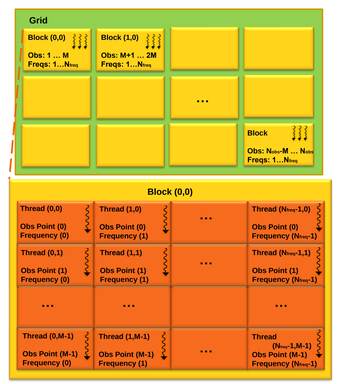


The induced surface current from each hit point needs to be radiated to every observation point in order to find the scattered field contributions.

- A 2D grid of blocks is allocated (2D is to avoid grid dimensional limit).
- Each block in the grid computes the **scattered field** at **M** **observation points** for all N_{freq} frequencies.
- The exact number of observation points per block is based on the amount of shared memory available per block.
- For a single frequency simulation this is around **290 observation points per block**. More frequencies require more memory and therefore allow fewer observation points per block.
- Each thread within the block (maximum of 512) computes the induced current and scattered field for one combination of frequency and observation point.
- This requires the host calling kernels sequentially for each hit point, looping through each hit point to accumulate the field values at all observation points in device global memory to be used by the next step.

4. Radiation from source to all N_{obs} observation points

The incident field from the source needs to be computed at every observation point in order to determine the total field.

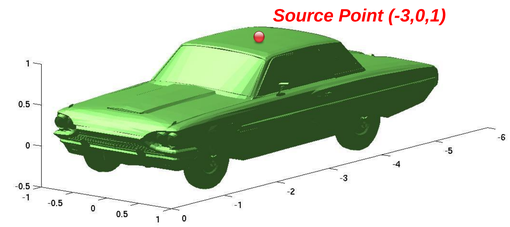


- A 2D grid of blocks is allocated (2D is to avoid grid dimensional limit).
- Each block in the grid computes the **incident field** at **M** **observation points** for all N_{freq} frequencies.
- The exact number of observation points per block is based on the amount of shared memory available per block.
- For a single frequency simulation this is around **290 observation points per block**. More frequencies require more memory and therefore allow fewer observation points per block.
- Each thread within the block (maximum of 512) computes the induced current and incident field for one combination of frequency and observation point.
- The field values for all frequencies and all observation points are added to the scattered field values in device global memory before transferring the result back to the host machine for visualization.

Results

Both versions of the CUDA™ implementation of the PO-SBR algorithm applied in this study gave roughly the same performance. When analyzing a multiple source simulation the GPU has reached speeds over 150x faster than the CPU alone (see HPEC 2010 presentation). After running the code and returning all field values to the host machine, MATLAB® was used to load and plot the results. Shown below are examples where the magnitude of the electric field (in dB) is plotted at different resolutions corresponding to a single Hertzian dipole antenna source radiating at a single frequency of 2 GHz. The simulation time and speed-up factors listed are for the radiation calculations, not including the ray tracing time which was not a significant part of the total run time for these cases. It can be seen that the speed-up over CPU is not very noticeable for small amounts of observation points because the GPU is not being fully utilized. But for large amounts of observation points here it grows to a factor of 50x.

1964 Thunderbird

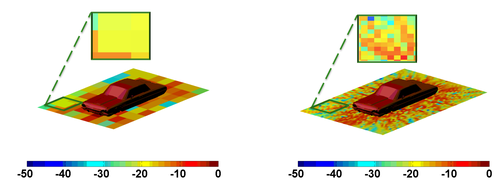


Timing Comparison

N_{obs}	CPU (sec)	GPU (sec)	Speedup
100	1.79	0.95	**1.88**
2,500	41.09	1.31	**31.37**
10,000	163.40	3.78	**43.23**
250,000	3,917.41	75.88	**51.63**

Field Profile

- 100 Observation Points
- 2,500 Observation Points



- 10,000 Observation Points
- 250,000 Observation Points