*Progress In Electromagnetics Research, Vol. 107, 187–202, 2010*

# FAST RCS PREDICTION USING MULTIRESOLUTION SHOOTING AND BOUNCING RAY METHOD ON THE GPU

**P. C. Gao, Y. B. Tao, and H. Lin**

State Key Laboratory of CAD&CG
Zhejiang University
Hangzhou 310058, China

**Abstract**—This paper presents a GPU-based multiresolution shooting and bouncing ray (MSBR) method with the kd-tree acceleration structure for the fast radar cross section (RCS) prediction of electrically large and complex targets. The multiresolution grid algorithm can greatly reduce the total number of ray tubes, as it adaptively adjusts the density of ray tubes for regions with different complexities of their structures, while the kd-tree acceleration structure can highly decrease the number of ray-patch intersection tests. The multiresolution grid technique and kd-tree traversal algorithm are fully implemented on the GPU to further accelerate the SBR by exploiting the massively parallel computing ability. Numerical experiments demonstrate that the proposed GPU-based MSBR can significantly improve the computational efficiency. It is about 40 times faster than the CPU MSBR, and at least 4.8 times faster than the GPU-based SBR without the multiresolution grid algorithm.

## 1. INTRODUCTION

The high-frequency method has various applications, for instance, the prediction of radar cross section (RCS), electromagnetic scattering [1], and automatic target recognition [2], and the shooting and bouncing ray (SBR) method [3] is one of the most popular and novel high-frequency methods. The SBR involves two parts: ray tube tracing and electromagnetic computing. Firstly, dense ray tubes, which represent an incident plane wave, are shot to the target, and these ray tubes are recursively traced to obtain the intersection positions. Then, the

*Received 18 June 2010, Accepted 22 July 2010, Scheduled 5 August 2010*
Corresponding author: H. Lin (lin@cad.zju.edu.cn).

central rays of ray tubes are also launched along the incident direction to the target, and the reflected fields of ray tubes are calculated according to the geometric optics (GO) during each reflection. Finally, the physical optics (PO) integral is carried out on the exit apertures, which are formed by the exit points of ray tubes, to obtain the scattered field. The feature of this approach is that it models the physical procedure clearly and is very simple in concept to implement.

In the SBR, the density of ray tubes on the virtual aperture perpendicular to the incident direction should be greater than ten rays per wavelength to ensure the convergence of results. This requirement makes it very time-consuming for electrically large targets. Many efforts have been made to develop various acceleration algorithms. One category is to reduce the number of ray-patch intersection tests. Jin et al. [4] used the octree, which is constructed by recursively subdividing the box into eight children boxes, to decrease the number of ray-patch intersection tests. Tao et al. [5] proposed to apply the kd-tree to reduce the time needed to trace each ray tube, since the kd-tree has been publicly known as the best acceleration structure for ray tracing of the static scenes in computer graphics [6]. The other category is to reduce the total number of ray tubes involved in the computation. Suk et al. [7] presented a fast algorithm to accelerate the SBR by applying the multiresolution grid algorithm. Large-size ray tubes are used to reduce the total number of ray tubes, and ray tubes are subdivided only when necessary. Bang et al. [8] combined the multiresolution grid subdivision algorithm and the octree structure to further improve the efficiency of the SBR. Recently, the angular division algorithm was developed to accelerate the SBR [9, 10].

Over the past few years, graphics processing unit (GPU) is changed from the special graphics hardware to the general computing resources, a highly parallel manycore processor with tremendous computational horsepower and very high memory bandwidth. Beside rendering, the GPU has been successfully applied to other complex computational problems, which is known as the general purpose processing on the GPU (GPGPU) [11]. Especially, the compute unified device architecture (CUDA) [12] developed by NVIDIA provides a simple and efficient way to leverage the massively parallel resources on the GPU. Many works on computational electromagnetics have been reported to use the GPU for acceleration. The Z-buffer was used in the graphical electromagnetic computing (GRECO) method [13], and it was improved for fast RCS prediction [14]. Tao et al. [15] accelerated the GRECO method by moving the electromagnetic computing part to the graphics hardware and achieved 30 times faster results. Peng and Nie [16] proposed an approach to speed up the conventional method

of moments on the GPU, which can deal with electrically large targets by splitting a huge matrix into sub matrixes. The finite-difference frequency-domain (FDFD) and the unconditionally stable Crank-Nicolson time-domain finite-difference (CN-FDTD) method were also implemented on the GPU for acceleration [17, 18].

We combine the kd-tree with the multiresolution grid algorithm to speed up the SBR, and simultaneously adopt the GPU as hardware acceleration due to its highly parallel architecture. Tao et al. [19] proposed a GPU-based SBR technique and achieved an acceleration ratio about 30. However, only the kd-tree is used in their approach to improve the efficiency of ray tube tracing. Our GPU-based MSBR further utilizes the multiresolution grid algorithm to reduce the number of ray tubes involved in the computation on the GPU. The rest of this paper is organized as follows. Section 2 introduces the conventional SBR and two main acceleration techniques which are the kd-tree acceleration structure and the multiresolution grid algorithm. Section 3 first gives a short overview of CUDA, and then presents the implementation detail of the multiresolution SBR (MSBR) with the kd-tree on the GPU. In Section 4, numerical results are presented with discussion. At last, Section 5 concludes this proposed approach.

## 2. SBR AND TWO ACCELERATION ALGORITHMS

Ray tube tracing and electromagnetic computing are two steps of the SBR. In the first step, a set of dense ray tubes are shot to the target along the incident direction, and ray tubes are traced to obtain the intersections with the target. Then, the central rays of ray tubes are launched, and the reflected fields of them are calculated by the theory of GO at each intersection as follows:

$$\begin{bmatrix} E_{\parallel}^{refl} \\ E_{\perp}^{refl} \end{bmatrix} = \begin{bmatrix} \Gamma_{\parallel} & 0 \\ 0 & \Gamma_{\perp} \end{bmatrix} \begin{bmatrix} E_{\parallel}^{inc} \\ E_{\perp}^{inc} \end{bmatrix} \tag{1}$$

where

$$\begin{aligned} E_{\parallel}^{inc} &= \hat{\mathbf{e}}_{\parallel} \cdot \mathbf{E}^{inc} \\ E_{\perp}^{inc} &= \hat{\mathbf{e}}_{\perp} \cdot \mathbf{E}^{inc} \\ \hat{\mathbf{e}}_{\perp} &= \hat{\mathbf{k}}^{inc} \times \hat{\mathbf{n}} / \left| \hat{\mathbf{k}}^{inc} \times \hat{\mathbf{n}} \right| \\ \hat{\mathbf{e}}_{\parallel} &= \hat{\mathbf{k}}^{inc} \times \hat{\mathbf{e}}_{\perp} / \left| \hat{\mathbf{k}}^{inc} \times \hat{\mathbf{e}}_{\perp} \right| \\ \hat{\mathbf{e}}_{\perp}^{refl} &= \hat{\mathbf{e}}_{\perp} \\ \hat{\mathbf{e}}_{\parallel}^{refl} &= \hat{\mathbf{k}}^{refl} \times \hat{\mathbf{e}}_{\perp}^{refl} / \left| \hat{\mathbf{k}}^{refl} \times \hat{\mathbf{e}}_{\perp}^{refl} \right| \end{aligned} \tag{2}$$

The vector $\hat{\mathbf{k}}^{inc}$ is the incident direction, $\hat{\mathbf{k}}^{refl}$ is the propagation direction after the reflection, and $\hat{\mathbf{n}}$ is the normal of the patch where the bounce happens. With the known $\mathbf{E}^{inc}$ and the parameters calculated above, the reflected field is:

$$\mathbf{E}^{refl} = \hat{\mathbf{e}}_{\parallel}^{refl} \cdot E_{\parallel}^{refl} + \hat{\mathbf{e}}_{\perp}^{refl} \cdot E_{\perp}^{refl} \tag{3}$$

The reflection coefficients are discussed in detail in [3, 20]. In the step of electromagnetic computing, the final PO integral is carried out with the exit field of the central ray on the four-sided polygon modeled by the exit positions of the ray tube. At an observation point $(r, \theta, \phi)$, the formula of the scattered field of a ray tube is given:

$$\mathbf{E}(r, \theta, \phi) \approx \frac{e^{-jkr}}{r}(\hat{\theta}E_{\theta} + \hat{\phi}E_{\phi}) \tag{4}$$

The PO integral about the aperture field $(\mathbf{E}, \mathbf{H})$ on the four-sided polygon $S$ is expressed as:

$$\begin{bmatrix} E_{\theta} \\ E_{\phi} \end{bmatrix} = \left(\frac{jk}{2\pi}\right) \iint_S e^{j\mathbf{k}\cdot\mathbf{r}'} \left\{ \begin{bmatrix} -\hat{\phi} \\ \hat{\theta} \end{bmatrix} \times \mathbf{E}(r') f_e + Z_0 \begin{bmatrix} \hat{\theta} \\ \hat{\phi} \end{bmatrix} \times \mathbf{H}(r') f_h \right\} \cdot \hat{\mathbf{n}} dx' dy'. \tag{5}$$

Equation (5) corresponds to three formulas with different values of the coefficients $f_e$ and $f_h$. The $EH$ formula ($f_e = f_h = 0.5$) gives the best approximation for the PO induced currents [21].

### 2.1. Kd-tree Construction and Traversal

The kd-tree, a well-known space-partition data structure, is widely used to accelerate ray tracing in computer graphics. Recently, it has been successfully applied in the application of computational electromagnetics [5]. A kd-tree is constructed by recursively splitting the target space into uneven boxes. The splitting plane is perpendicular to one of three coordinate axes, and the greedy surface area heuristic (SAH) strategy [22] is commonly used to search the optimal splitting plane. A kd-tree has two kinds of nodes: the interior node and the leaf node. The interior node has two children nodes and the leaf node contains patches that overlap the axis-aligned box of the leaf node. The depth-first kd-tree traversal algorithm needs a stack to preserve the to-be-visited nodes. Due to the lack of the stack on the GPU, a stackless kd-tree traversal algorithm was developed by augmenting the kd-tree with ropes [23]. These ropes link the leaf node via its six faces directly to the adjacent leaf node, the smallest interior node or null.

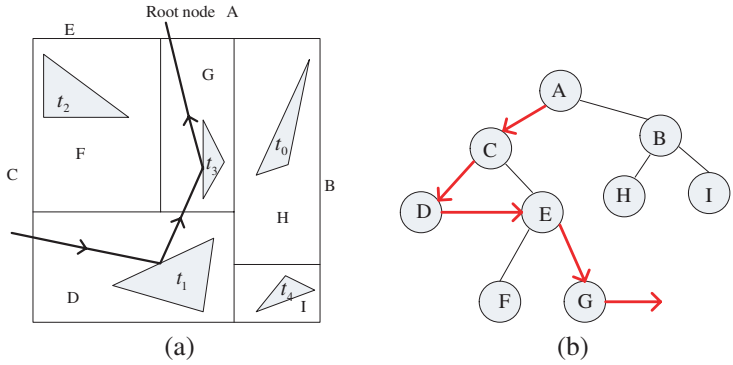


**Figure 1.** A 2D kd-tree and its traversal. (a) A ray has two intersections with the 2D kd-tree, which has the interior nodes (B, C, E) and the leaf nodes (D, F, G, H, I). (b) The traversal path is marked with a red bold line. The ray enters the root node A, continues the traversal through the interior node C. Then an intersection is found in the leaf node D. With the help of the rope of D, the generated reflected ray directly moves to the smallest interior node E including the leaf node F and G. Ray tracing ends with the second reflected ray exiting the leaf node G.

Figure 1 shows the traversal procedure of a 2D kd-tree with ropes. When the ray encounters an interior node, one children node of the interior node is selected to be traversed according to the relative position of the splitting plane and the position where the ray enters the node. At the leaf node, the intersection tests between the ray and patches are performed to find the intersection position. If the ray has no intersection with patches of the leaf node, it continues the traversal through the rope efficiently. In this way, the stackless kd-tree traversal greatly reduces the number of the interior node traversal steps to save the ray tracing time in the SBR.

## 2.2. Multiresolution Grid Algorithm

In the SBR, for accurate results, the density of the incident rays on the virtual plane should be greater than ten rays per wavelength. A large number of rays will be the computational burden with the growth of the electrical size of the target. Thus, an adaptive and flexible approach called the multiresolution grid algorithm was developed to overcome this problem [7]. The multiresolution grid algorithm first uses large-size ray tubes, and then each ray tube may be evaluated to obtain the scattered field, or be subdivided into four children ray tubes according to the divergence.

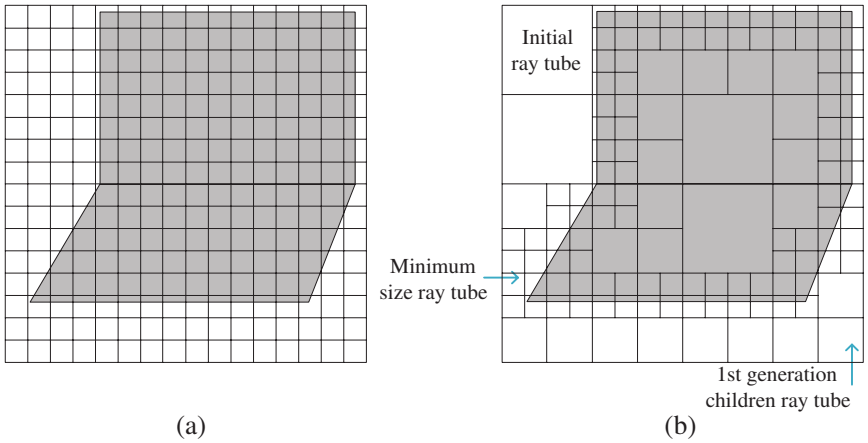


**Figure 2.** Ray tube grid on the virtual plane. (a) The fixed resolution grid. (b) The multiresolution grid.

Figure 2 compares the virtual aperture partition of the SBR and the multiresolution SBR in predicting the first order scattering. The target is formed by two patches, and the density of ray tubes is $16 \times 16$. In the SBR, there are 289 corner rays that need to be traced. However, only 175 rays need to be traced in the multiresolution grid algorithm with the same accuracy.

## 3. GPU-BASED MSBR WITH THE KD-TREE

The current GPU architecture is based on a scalable array of streaming multiprocessors (SMs) and a hierarchy of memory mainly including on-chip shared memory and global memory located out of the chip. CUDA is a minimal extension of the C programming language, and the core of the parallel programming model of CUDA is the abstraction: a hierarchy of thread groups, which provides a simple and direct way to access the massively parallel computing resources on the GPU. As shown in Figure 3, the hierarchy includes three concepts: grid, block, and thread. A grid that consists of several thread blocks corresponds to a parallel task. The threads within the same block can access the same shared memory to cooperate among themselves. The global memory is used to communicate and share large data sets among different tasks. More introduction about CUDA can be found in [12].

The procedure of the GPU-based MSBR is shown in Figure 4, and it executes in the multipass manner. One pass deals with ray tubes at the fixed size, and includes ray tube tracing, divergent ray tube recording, and electromagnetic computing of the valid ray tubes. The

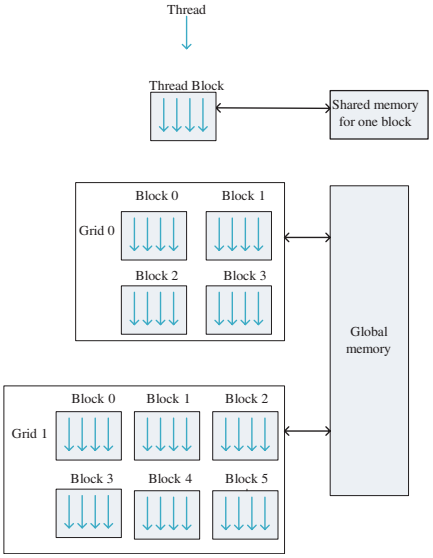


**Figure 3.** The hierarchic architecture of CUDA.

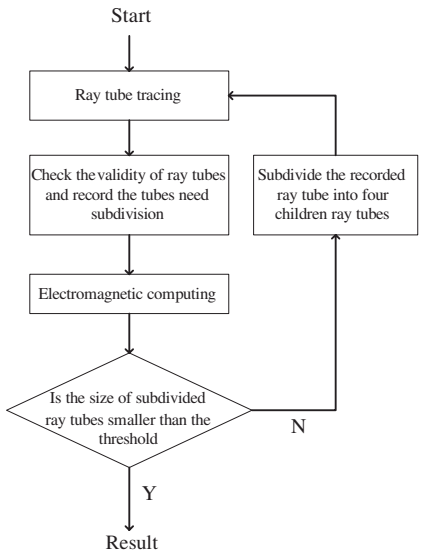


**Figure 4.** The procedure of the GPU-based MSBR.

next pass subdivides divergent ray tubes and continues to process these generated children ray tubes until the ray tube size is smaller than the user defined threshold. Figure 5 illustrates ray tube grid on the virtual plane, and this grid is subdivided into 2D thread blocks. Each thread of the block traces a corner ray to obtain its intersection information in parallel. In the first step, the stackless kd-tree traversal algorithm (Section 2.1) is used to accelerate ray tube tracing. The final step, electromagnetic computing, employs the GO and PO to evaluate the scattered fields of ray tubes (Section 2), and these results are summed up to obtain the scattered field of the target in this pass. The second step is to check the validity of ray tubes and record divergent ray tubes, described in detail as follows.

As shown in Figure 6, the validity of ray tubes is checked according to the four cases of relationships between the ray tube and patches of the target. We abandon a ray tube belonged to the type A which misses the target. A ray tube is designated as the type B when four corner rays all intersect with the same patch. The type C is that four corner rays bounce from adjacent patches with the similar normal. A ray tube belongs to the type D when it intersects partially with the target or bounces from the patches which have largely different normals. According to the above rules, ray tubes of the type B

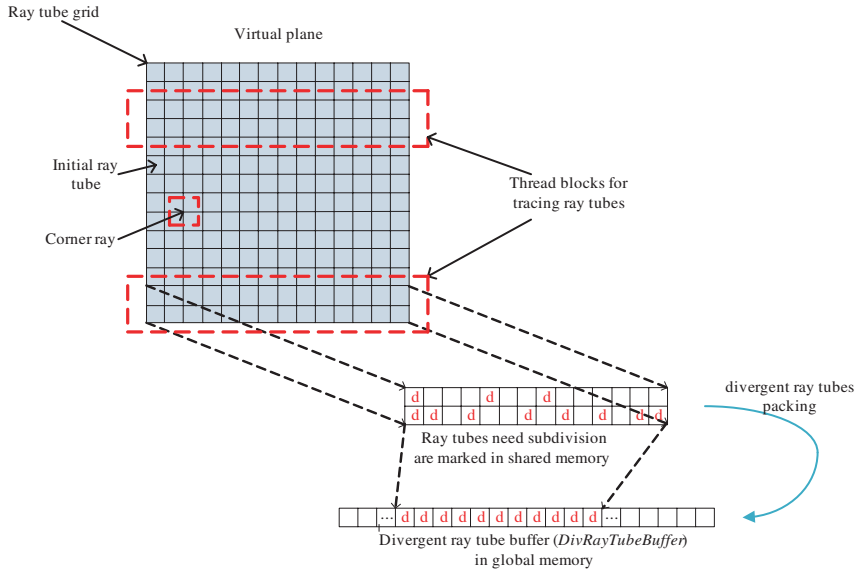


**Figure 5.** The procedure of divergent ray tubes recording on the GPU.

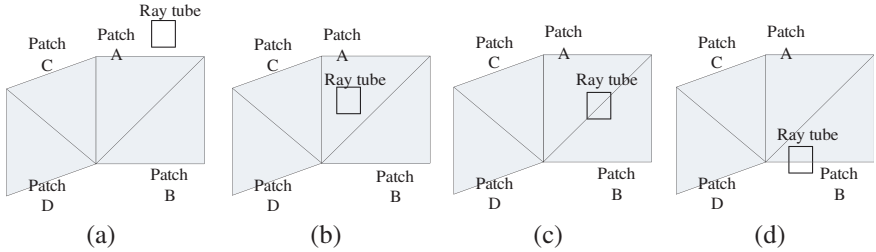


**Figure 6.** Four types ray tubes. (a) Type A. (b) Type B. (c) Type C. (d) Type D.

and C will be processed in the step of electromagnetic computing to obtain the scattered fields, while ray tubes of the type D are divergent ray tubes and should be recorded in the divergent ray tube buffer (*DivRayTubeBuffer*) in global memory on the GPU.

The divergent information of each ray tube is first recorded in the shared memory of the block, as indicated with "d" in Figure 5. Since divergent ray tubes are located sparsely on the ray tube grid and global memory is limited on the GPU, it is necessary to pack them in *DivRayTubeBuffer* to reduce the total memory usage [24]. The first thread of each block is chosen to count the number of divergent ray

tubes and write them to *DivRayTubeBuffer*. Threads of different blocks may access the same address of global memory causing the write-write conflict when they perform the write operation. This is due to the parallel execution of threads in CUDA. Fortunately, the atomic add function performing a read-modify-write atomic operation provided by CUDA can solve the write conflicts. It reads an integer in global memory, adds another integer to it, and writes the result to the same address. Figure 7 shows the application of the atomic add operation in the MSBR. The grid keeps an integer which records the current number of ray tubes in *DivRayTubeBuffer*, and each block can find its exact position to write.

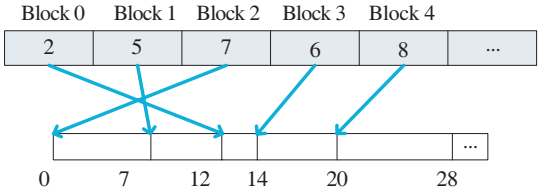


**Figure 7.** As blocks execute in parallel, the starting position of each block in *DivRayTubeBuffer* is random. Here, we assume that blocks perform the atomic add operation in order 2, 1, 0, 3, 4, ....

It should be noted that it may seem inefficient for the first thread to perform the write operation while the others in the same block are waiting. A possible way is to let each thread corresponding to a divergent ray tube to write its own content to *DivRayTubeBuffer*. For this solution, one thread should sum the number of the divergent ray tubes before it in shared memory to locate its own position. However, no performance improvement is shown in this method.

In the next pass, divergent ray tubes in the last pass are recorded in *DivRayTubeBuffer*, and each divergent ray tube should be subdivided into four children ray tubes. Corner rays of children ray tubes may have been traced in previous passes, and we can trace them again in this pass or copy and use the existing intersection information of them. A runtime analysis shows that the copy solution is the better choice.

In summary, our algorithm performs ray tube tracing, divergent ray tubes recording, and electromagnetic computing in one pass. Then, each ray tube in *DivRayTubeBuffer* is subdivided into four children ray tubes, and all generated children ray tubes execute the same three steps in the next pass. The subdivision of ray tubes repeats until the ray tube size is smaller than the user defined threshold, and then we get the final scattered field of the target.

## 4. NUMERICAL RESULTS AND DISCUSSION

Several numerical results are presented to evaluate the proposed GPU-based MSBR with the kd-tree acceleration structure. All the results are derived on an Intel Core 2 Duo 3.0 GHz CPU and a NVIDIA GeForce 275 GTX with CUDA Toolkit 2.2. Three different types of the targets, two ships and an airplane as shown in Figure 8, are tested to demonstrate the efficiency and accuracy of our approach. The detailed geometric information of the targets are listed in Table 1.

For all the experiments described in the following, the incident direction of the ship A and the airplane is rotated around the $Y$ axis from 0° to 360° with the interval of 1°, while the $\phi$ is fixed at 0°. The incident direction for the ship B is rotated around the $Z$ axis from 0° to 360° with the interval of 1°. At most fifth-order is considered for the ship B and the airplane, while only second-order is used for the ship A due to its simple structure. The initial size of ray tubes is set $0.8\lambda$, and the threshold is $0.1\lambda$ in both the CPU and GPU MSBR. The GPU-based SBR adopts $0.1\lambda$ as the size of ray tubes.

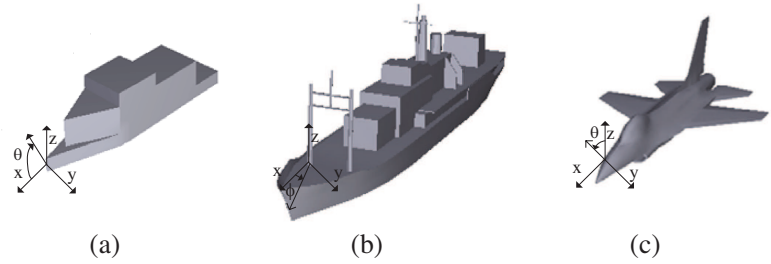


**Figure 8.** Three targets: (a) Ship A. (b) Ship B. (c) Airplane.

**Table 1.** The geometry size and the number of triangles of the three targets. The peak memory requirement of the ship A, the ship B and the airplane at 1 THz, 10 GHz and 15 GHz for the GPU-based MSBR.

| Target | Size(m) | Triangle Number | Peak Memory Usage (M) |
|---|---|---|---|
| Ship A | $0.9 \times 0.2 \times 0.2$ | 574 | 101.6 |
| Ship B | $43.71 \times 5.89 \times 9.144$ | 3237 | 271.8 |
| Airplane | $11.76 \times 7.4 \times 3.67$ | 13050 | 278.9 |

**Table 2.** The computation time (second) comparison of our GPU-based MSBR, the CPU MSBR and the GPU-based SBR [19] for three targets.

| Method | Ship A | Ship B | Airplane |
|---|---|---|---|
| MSBR on the CPU | 10575.3 | 3381.7 | 3958.2 |
| GPU-based SBR | 1209.2 | 442 | 463.1 |
| GPU-based MSBR | 44.1 | 79.4 | 96.2 |
| Acceleration ratio(CPU \ GPU) | 239.8 \ 27.4 | 42.6 \ 5.6 | 41.1 \ 4.8 |

The monostatic RCS of the ship A, the ship B and the airplane were computed at 1 THz, 10 GHz, and 15 GHz, respectively. The computational time of three methods and the acceleration ratio of each target are shown in Table 2, and the peak memory requirement of the three targets is listed in Table 1 for the GPU-based MSBR.

The GPU-based MSBR is at least 40 times faster than the CPU MSBR for the three targets due to the strongly parallel computing power of the GPU. The proposed technique also greatly improves the performance of the GPU-based SBR [19] at least 4.8 times. We also perform other tests to demonstrate the potential of our algorithm for extremely electrically large targets. For the ship A in the Terahertz application, our approach is 239.8 and 27.4 times faster than the corresponding CPU and GPU methods, respectively. Although the speedup ratio decreases to about 5 compared with the GPU-based SBR, our method is more suitable for extremely electrically large targets. For example, the speedup ratio reaches up to 10.7 for the airplane illustrated in Figure 8(c) at 100 GHz. As the frequency increases, the electrical size of the target also grows, and the proposed approach can achieve a higher acceleration ratio in this situation.

The number of ray tubes on the virtual plane is proportional to the projected area of the target and the frequency. The maximum number of the initial ray tubes is $3304 \times 5126$ for the airplane illustrated in Figure 8(c) at 100 GHz when the initial size of ray tubes is $0.8\lambda$. Each corner ray needs to record the intersections with the target and the scattered field of each ray tube requires 8 float number. In this situation, 1873 M memory is needed for the initial resolution and more memory will be allocated for divergent ray tubes. However, the amount of available memory on the GPU is limited, for instance, NVIDIA GeForce 275 used in this paper has only 896 M memory. As the proposed method should be scalable with respect to targets with different electrical sizes on various GPU cards, we partition the virtual

plane into sub-grids. The scattered field of each sub-grid is calculated by the GPU-based MSBR, and these scattered fields are summed up to obtain the total scattered field of the target. In our implementation, the sub-grid size is $1024 \times 1024$. The ray tubes need no subdivision in the GPU-based SBR, since the size of ray tubes is fixed. Thus, the memory requirement of the GPU-based SBR is at most 116 M for considering fifth-order reflection corresponding to the sub-grid size of $1024 \times 1024$. However, the GPU-based MSBR need additional memory for recording the intersection information and scattered fields of the divergent ray tubes. The peak memory requirement of the GPU-based MSBR is shown in Table 1, and the largest memory requirement of the GPU-based MSBR is 278.9 M in our experiments.

The ship A illustrated in Figure 8(a) is widely used as the benchmark to validate the accuracy of the SBR [4, 7, 8]. The GPU-based MSBR result of the monostatic RCS of the ship A at 10 GHz is compared with the result calculated with the MLFMM [25, 26] in Figure 9. The result of our GPU-based MSBR agrees well with the MLFMM result, and the deviation maybe partly due to lack of the edge-diffraction effect [27]. We add the diffraction field of the truncated-wedge incremental-length diffraction coefficients (TW-ILDC) [28] to the scattered field of the GPU-based MSBR, and compare it with the MLFMM result. The comparison result illustrated in Figure 9 confirms our conjecture. The computational time of the GPU-based MSBR and TW-ILDC are 3.3 seconds and 2.6 seconds, respectively. Although the TW-ILDC would reduce the computational efficiency of the proposed method, about 1.8 times in this case, the TW-ILDC can greatly improve the accuracy of the RCS prediction.

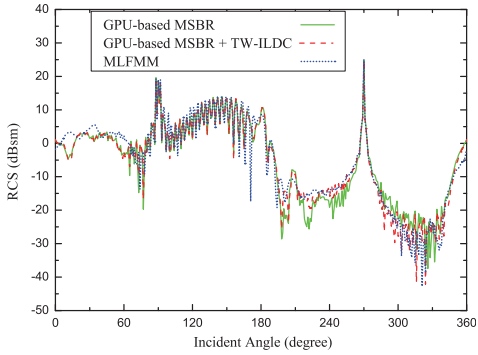


**Figure 9.** The comparison of our GPU-based MSBR result, our GPU-based MSBR + TW-ILDC result and the MLFMM result for the ship A at 10 GHz in vv-polarization.

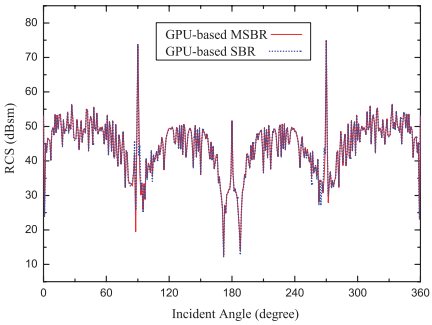


**Figure 10.** The comparison of our GPU-based MSBR result and the GPU-based SBR result for the ship B at 10 GHz in vv-polarization.

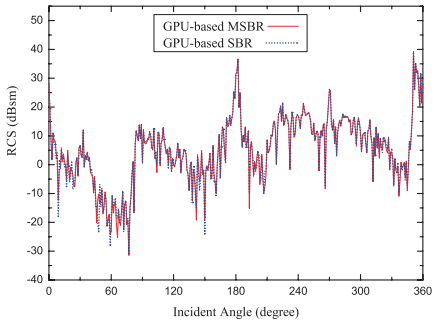


**Figure 11.** The comparison of our GPU-based MSBR result and the GPU-based SBR result for the airplane at 15 GHz in vv-polarization.

The total computational time of the GPU-based MSBR + TW-ILDC is 5.9 seconds, and it is much faster than the MLFMM, 28 hours for the vv result. The peak memory requirement of the ship A at 10 GHz is 68.7 M, and it is less than 388.6 M of the MLFMM again. We also compare the results of the ship B and the airplane calculated by our GPU-based MSBR with the results of the GPU-based SBR [19]. Again, good agreements are observed both in Figure 10 and Figure 11.

## 5. CONCLUSION

In this paper, the multiresolution grid algorithm and the kd-tree acceleration structure are combined and implemented on the GPU in order to improve the computational efficiency of the SBR. The multiresolution grid algorithm highly reduces the total number of ray tubes involved in the computation by avoiding the unnecessary ray tube partition for regions without complex structures. Moreover, the kd-tree acceleration structure is used to reduce the tracing time for a single ray tube. Compared with both the CPU MSBR and the GPU-based SBR [19], the proposed approach provides a more efficient solution for the RCS prediction of extremely electrically large and complex targets. Apparently, adopting the GPU in electromagnetic computing to exploit its powerfully parallel computing ability is a new trend. With the rapid development of graphics hardware, more efforts should be made to accelerate the existing approaches with the GPU and develop new advanced algorithms adapting to parallel hardware architectures simultaneously.

## ACKNOWLEDGMENT

We would like to thank Prof. T. Cui from South East University for providing the MLFMM method used in this paper.

## REFERENCES

1. Li, X.-F., Y.-J. Xie, and R. Yang, "High-frequency method analysis on scattering from homogenous dielectric objects with electrically large size in half space," *Progress In Electromagnetics Research B*, Vol. 1, 177–188, 2008.
2. Park, S. H., K. K. Park, J. H. Jung, H. T. Kim, and K. T. Kim, "Construction of training database based on high frequency RCS prediction methods for ATR," *Journal of Electromagnetic Waves and Applications*, Vol. 22, No. 5–6, 693–703, 2008.
3. Ling, H., R. C. Chou, and S. W. Lee, "Shooting and bouncing rays: Calculating the RCS of an arbitrarily shaped cavity," *IEEE Trans. Antennas Propag.*, Vol. 37, No. 2, 194–205, 1989.
4. Jin, K. S., T. I. Suh, S. H. Suk, B. C. Kim, and H. T. Kim, "Fast ray tracing using a space-division algorithm for RCS prediction," *Journal of Electromagnetic Waves and Applications*, Vol. 20, No. 1, 119–126, 2006.
5. Tao, Y.-B., H. Lin, and H. J. Bao, "Kd-tree based fast ray tracing for RCS prediction," *Progress In Electromagnetics Research*, Vol. 81, 329–341, 2008.
6. Havran, V., "Heuristic ray shooting algorithms," Ph.D. dissertation, Univ. Czech Technical, Prague, 2000.
7. Suk, S. H., T. I. Seo, H. S. Park, and H. T. Kim, "Multiresolution grid algorithm in the SBR and its application to the RCS calculation," *Microw. Opt. Technol. Lett*, Vol. 29, No. 6, 394–397, 2001.
8. Bang, J. K., B. C. Kim, S. H. Suk, K. S. Jin, and H. T. Kim, "Time consumption reduction of ray tracing for RCS prediction using efficient grid division and space division algorithms," *Journal of Electromagnetic Waves and Applications*, Vol. 21, No. 6, 829–840, 2007.
9. Kim, B.-C., K. K. Park, and H.-T. Kim, "Efficient RCS prediction method using angular division algorithm," *Journal of Electromagnetic Waves and Applications*, Vol. 23, No. 1, 65–74, 2009.
10. Park, K.-K. and H.-T. Kim, "RCS prediction acceleration and reduction of table size for the angular division algorithm," *Journal*

*of Electromagnetic Waves and Applications*, Vol. 23, No. 11–12, 1657–1664, 2009.

11. Owens, J. D., D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," *Comput. Graphics Forum*, Vol. 26, No. 1, 80–113, 2007.
12. NVIDIA Corporation, *NVIDIA CUDA Programming Guide 2.2.1*, http://developer.download.nvidia.com/compute/ cuda/2_21/toolkit/docs/NVIDIA_CUDA_Programming_Guide_ 2.2.1.pdf
13. Rius, J. M., M. Ferrando, and L. Jofre, "High frequency RCS of complex radar targets in real time," *IEEE Trans. Antennas Propag.*, Vol. 41, No. 9, 1308–1318, 1993.
14. Zha, F.-T., S.-X. Gong, Y.-X. Xu, Y. Guan, and W. Jiang, "Fast shadowing technique for electrically large targets using Z-buffer," *Journal of Electromagnetic Waves and Applications*, Vol. 23, No. 2–3, 341–349, 2009.
15. Tao, Y. B., H. Lin, and H. J. Bao, "From CPU to GPU: GPU-based electromagnetic computing (GPUECO)," *Progress In Electromagnetics Research*, Vol. 81 1–19, 2008.
16. Peng, S. X. and Z. P. Nie, "Acceleration of the method of moments calculations by using graphics processing units," *IEEE Trans. Antennas Propag.*, Vol. 56, No. 7, 2130–2133, 2008.
17. Zainud-Deen, S. H., E. El-Deen, M. S. Ibrahim, K. H. Awadalla, and A. Z. Botros, "Electromagnetic scattering using GPU-based finite difference frequency domain method," *Progress In Electromagnetics Research B*, Vol. 16, 351–369, 2009.
18. Xu, K., Z. H. Fan, D. Z. Ding, and R. S. Chen, "GPU accelerated unconditionally stable Crank-Nicolson FDTD method for the analysis of three-dimensional microwave circuits," *Progress In Electromagnetics Research*, Vol. 102, 381–395, 2010.
19. Tao, Y. B., H. Lin, and H. J. Bao, "GPU-based shooting and bouncing ray method for fast RCS prediction," *IEEE Trans. Antennas Propag.*, Vol. 58, No. 2, 494–502, 2010.
20. Balanis, C. A., *Advanced Engineering Electromagnetics*, Wiley, New York, 1989.
21. Baldauf, J., S. W. Lee, L. Lin, S. K. Jeng, S. M. Scarborough, and C. L. Yu, "High frequency scattering from trihedral corner reflectors and other benchmark targets: SBR vs. experiments," *IEEE Trans. Antennas Propag.*, Vol. 39, No. 9, 1345–1351, 1991.
22. Goldsmith, J. and J. Salmon, "Automatic creation of object hierarchies for ray tracing," *IEEE Computer Graphics and*

*Applications*, Vol. 7, No. 5, 14–20, 1989.

23. Popov, S., J. Günther, H. P. Seidel, and P. Slusallek, "Stackless kd-tree traversal for high performance GPU ray tracing," *Comput. Graphics Forum*, Vol. 26, No. 3, 415–424, 2007.
24. Jin, B., I. Ihm, B. Chang, C. Park, W. Lee, and S. Jung, "Selective and adaptive supersampling for real-time ray tracing," *Proceedings of the Conference on High Performance Graphics 2009*, Vol. 34, No. 10, 1064–1076, 2009.
25. Song, J. M. and W. C. Chew, "Multilevel fast multipole algorithm for solving combined field integral equations of electromagnetic scattering," *Microw. Opt. Tech. Lett.*, Vol. 10, 14–19, 1995.
26. Yang, M.-L. and X.-Q. Sheng, "Parallel high-order FE-BI-MLFMA for scattering by large and deep coated cavities loaded with obstacles," *Journal of Electromagnetic Waves and Applications*, Vol. 23, No. 13, 1813–1823, 2009.
27. Koujoumijan, R. G. and P. H. Pathak, "A uniform geometrical theory of diffraction for an edge in a perfectly conducting surface," *Proc. IEEE*, Vol. 62, 1448–1461, 1974.
28. Johansen, P. M., "Uniform physical theory of diffraction equivalent edge currents for truncated wedge strips," *IEEE Trans. Antennas Propag.*, Vol. 44, No. 7, 989–995, 1996.