Adaptive Aperture Partition in Shooting and Bouncing Ray Method

Yu Bo Tao, Hai Lin, and Hu Jun Bao

***Abstract*—The shooting and bouncing ray (SBR) method may give rise to a divergence problem when ray tubes intersect discontinuous parts of the target, such as the boundary, and this affects the accuracy to some extent. This paper proposes an adaptive aperture partition algorithm to solve this problem. The proposed algorithm adaptively splits the virtual aperture into continuous irregular beams instead of discrete uniform ray tubes according to the geometry of the target during the recursive beam tracing. These beams form a beam tree, the level of which represents the number of reflections. Geometric optics is applied to the representative propagation path of each leaf beam to obtain the exit field, and then physical optics is used to evaluate each leaf beam's scattered field. The proposed algorithm could generate the convergent solution of the SBR method when the ray-tube size tends toward infinitesimal. Additionally, this paper describes the beam-triangle intersection in detail and utilizes the kd-tree to accelerate the beam-target intersection. Numerical experiments demonstrate that adaptive aperture partition can greatly improve the accuracy of the SBR method, and the computational efficiency can be also significantly enhanced in several applications, such as the RCS prediction in the THz band.**

***Index Terms*—Adaptive aperture partition, beam tracing, beam-triangle intersection, kd-tree, radar cross section (RCS), shooting and bouncing ray (SBR).**

I. Introduction

THE prediction of the high-frequency scattering from arbitrarily shaped targets is of growing importance for the simulation of radar systems, such as the radar cross section (RCS) and inverse synthetic aperture radar (ISAR) applications. The shooting and bouncing ray (SBR) [1], [2] method is one of the principal ways to predict the scattered field of electrically large and complex targets with great accuracy and efficiency.

In the SBR method, the incident plane wave is described by means of a uniform grid of ray tubes, and the density of ray tubes should be greater than about ten rays per wavelength in view of the convergence of results. Four corner rays and the central ray of each ray tube are recursively traced to obtain the exit positions. The exit field of each ray tube is also traced and calculated during the central ray tracing according to the law of geometrical optics (GO) [3]. The field scattered from each ray tube can be evaluated through physical optics (PO), and the scattered field of the target is the sum of all scattered fields of ray tubes.

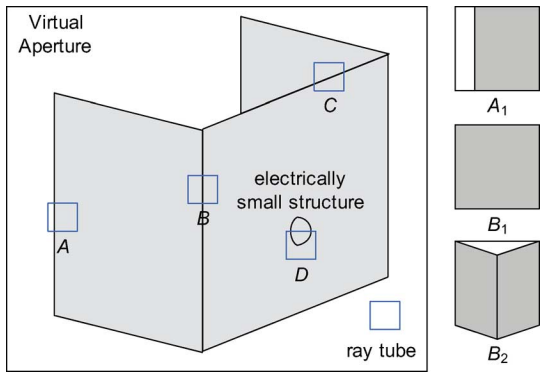

Fig. 1. The illustration of the ray-tube divergence problem. The target is three orthogonal rectangles with an electrically small structure on the middle rectangle, and its projection on the virtual aperture is the gray polygon. Two corner rays of the ray tube A do not intersect the target, the standard approach is to discard A. Actually, the intersected area of A is shown in the enlarged ray tube A_1 on the right. For the ray tube B, the standard approach uses the four intersection positions to construct an equivalent ray tube, as shown in the enlarged ray tube B_1. However, the intersected area of the ray tube B is shown in the enlarged ray tube B_2. It is more complex to process the divergent ray tubes C and D.

The independence of corner/central ray tracing makes the SBR method easily implementable and highly effective. However, it may result in a divergence problem, i.e., the ray tube diverges in the recursive ray tracing, as illustrated in Fig. 1. This is different from the divergence factor in the original SBR method [1], which describes the change in the field magnitude with the size of the ray tube when it intersects the target defined by the parametric surface. In this paper, the target is described in terms of triangles, and the divergent ray tube may partially intersect the target or may be reflected without one dominant propagation direction. Concretely, when any one of corner rays does not intersect the target, such as the ray tube A in Fig. 1, this ray tube is divergent. The current approach to processing divergent ray tubes is to discard these invalid ray tubes directly, not including them in the calculation of the scattered field. However, the simple approach would affect the accuracy of the SBR method, especially when the frequency is down to 500 MHz [4]. Although the high-frequency approximation would not be accurate enough in the low frequency, the divergence problem may be another factor for inaccurate results. This is because the size of ray tubes is so large enough that the scattered fields of these discarded ray tubes can not be simply ignored. More complexly, ray tubes would diverge in the recursive ray tracing, and it is very difficult to identify this divergence accurately due to the discrete sampling, such as the ray tube B, C, and D in Fig. 1. Even if

Manuscript received April 30, 2010; revised December 20, 2010; accepted January 15, 2011. Date of publication July 12, 2011; date of current version September 02, 2011. This work was supported in part by the National Hi-Tech Research and Development Program of China under Grant 2002AA135020.

The authors are with the State Key Laboratory of CAD&CG, Zhejiang University, Hangzhou 310058, China (e-mail: taoyubo@cad.zju.edu.cn; lin@cad.zju.edu.cn; bao@cad.zju.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TAP.2011.2161435

the equivalent cross section of the ray tube is constructed from the four intersection positions to calculate the scattered field, it is still different from the actual scattered field, especially for the ray tube C and D. Therefore, diverged ray tubes, whether it is identified or not, would affect the accuracy of the SBR method.

The ray-tube divergence problem arises from the procedure of the SBR method: first divide the grid into dense uniform ray tubes, and then trace each ray tube individually. This pre-partition procedure does not consider the geometry of the target. Therefore, ray tubes may intersect the discontinuous area of the target in the recursive ray tracing, such as the boundary and the electrically small and complex part, and this leads to the ray-tube divergence problem.

In this paper, we propose an adaptive aperture partition algorithm based on beam tracing to solve the divergence problem. The basic idea is to delay the virtual aperture partition until the recursive beam tracing and divide the aperture into beams only if necessary. A beam is a continuous volume of rays, while a ray tube is simplified to five discrete rays (four corner rays and one central ray) without taking into account the space in the ray tube. Beam tracing was first introduced by Heckbert and Hanrahan [5] in 1984. In computer graphics, beam tracing utilizes the geometric coherence of rays, i.e., neighbor rays usually intersect the same triangle of the target and have the same propagation path, to improve the efficiency of ray tracing. Beam tracing has been employed in antialiasing [6] and the calculation of exact soft shadows [7]. In acoustic modeling, beam tracing has been widely used to calculate exact sound propagation paths from the source to the receiver in virtual environments [8], [9].

Beam tracing has also been applied in the radio propagation prediction to find all exact propagation paths from a transmitter to a receiver. Son and Myung [10] presented a ray tube tree based on uniform geometrical theory of diffraction (UTD) for quasi 3D environments. An enhanced three-dimensional beam-tracing algorithm including diffraction phenomena has been developed by Bernardi *et al.* [11] to evaluate the field distribution in complex indoor environments. Recently, Kim and Lee [12] presented the concept of ray frustums, which is similar to the beam. They also introduced several acceleration techniques, such as quad tree, for fast ray frustums traversal in the environment. Compared to ray-tracing methods, these beam-tracing algorithms eliminate reception tests and existence tests of propagation paths, and improve the numerical efficiency and accuracy.

There have been several publications on how to partition the virtual aperture more effectively. Suk *et al.* [13] presented a multiresolution grid algorithm, which recursively subdivide the divergent ray tube into four uniform children ray tubes until the size of the divergent ray tube is less than the criterion. Although the multiresolution grid algorithm greatly reduces the initial number of ray tubes and accelerates the ray tracing of the SBR method, it still suffers from the ray-tube divergence problem, as the partition of the grid does not take into account the geometry of the target. Weinmann [14] introduced a random sampling strategy on the virtual aperture plane. For each sample, a ray is constructed and recursively traced, and a equivalent ray tube is generated at the intersection position to assess the scattered field. Although it is obvious that using a ray to represent a ray tube can avoid the divergence problem to a certain extent, more additional samples are required to reduce the prediction error. Recently, Xu and Jin [15] developed analytic tracing of polygonal ray tubes for precisely calculating the illumination and shadowing of triangles. However, they did not describe the 3D implementation in detail. For arbitrarily shaped targets, the effective implementation of adaptive aperture partition requires the exact understanding of the beam-triangle intersection. As a result, we systematically discuss the procedure of the beam-triangle intersection in this paper.

In addition, various acceleration techniques have been proposed to reduce the computational time of the SBR method, especially the ray tracing. Jin *et al.* [16] introduced the octree to decrease the number of ray-triangle intersection tests. The kd-tree has been proved as the best general-purpose acceleration structure for ray tracing of static scenes in computer graphics [17]. Therefore, Tao *et al.* [18] utilized the kd-tree to accelerate the ray tracing of the SBR method, and they extended this work to a GPU-based SBR method fully executed on the graphics processing unit (GPU) [19] for fast RCS prediction. In this paper, we also use the kd-tree to accelerate the beam-target intersection based on the work of a beam tracer [7]. The key difference is that the beam in the beam tracer is a polygonal pyramid emitting from a point source, while the beam here is a polygonal prism launched from the virtual aperture with the same direction. It is necessary, therefore, to adapt the existing beam tracing in computer graphics to simulate the plane wave.

This paper is organized as follows. We first introduce an overview of the proposed adaptive aperture partition in Section II. The beam-triangle intersection and kd-tree beam traversal are described in Section III, and this procedure generates a beam tree representing all possible propagation paths. Section IV is dedicated to the use of the beam tree to predict the scattered field of the target. The experimental results and discussions are covered in Section V. Finally, the conclusions are drawn in Section VI.

II. Method Overview

In the SBR method, the grid on the virtual aperture is divided into dense ray tubes uniformly before the ray tube tracing, and the ray tube is a square prism started from the virtual aperture. In this paper, a beam is a quadrangular prism or a triangular prism with an irregular cross-section, and a beam is marked as hit or miss depending on whether it intersects the target. The initial beam is launched from the virtual aperture, and the reflected beam is started from the intersected area on the hit triangle.

Adaptive aperture partition is performed during beam tracing and is described as follows. Once the incident direction of the electromagnetic wave has been specified, we first construct the polygonal virtual aperture perpendicular to the incident direction, which should be large enough to cover at least the projected area of the target. The whole aperture as an initial beam is traced in the space of the target. When the beam encounters the geometry of the target, it is dynamically split according to the projection of the triangle and generates several irregular hit beams and miss beams. The generated beam continues to be traced until it hits the nearest triangle or exits the target. When the intersection between the initial beam and the target is finished, the virtual

aperture has been adaptively partitioned and a group of primary beams are generated. For each hit beam, the intersected area on the hit triangle acts as the virtual aperture of the reflected beam. The reflected beam is recursively traced and its virtual aperture is also adaptively split until it exits the target or the number of intersections is larger than the maximum order of the reflection. The kd-tree can be used to accelerate the beam-target intersection. Finally, all hit beams form a beam tree.

With the beam tree, we can construct a representative propagation path for each leaf beam. The exit field of each leaf beam can be calculated by GO based on the representative propagation path, and the PO integral is applied to each leaf beam to evaluate the scattered field of the target.

As the splitting lines are these projected edges of triangles visible from the incident direction, each hit beam intersects only one triangle. Thus, the virtual aperture is divided according to the geometry of the target and adaptive aperture partition overcomes the divergence problem. The fast generation of the beam tree and electromagnetic computing based on the beam tree are detailed in the following sections.

III. Adaptive Aperture Partition

Adaptive aperture partition is actually the process of the beam-target intersection: the primary beams are the intersection result between the root beam and the target, while the secondary beams are generated through the intersection of the hit beam and the target. In order to describe the adaptive aperture partition algorithm clearly, we first introduce the basic beam-target intersection procedure, and then describe how to accelerate this intersection procedure using the kd-tree.

A. Beam-Target Intersection

Since the target in this paper is modeled by triangles, the fundamental operation of the beam-target intersection is the beam-triangle intersection. The beam-triangle intersection would split the beam into the hit part and the miss part, which are equal to the blocked part and the passed part of the beam. This intersection is similar to standard geometry set operations [5], [7], such as the intersection and difference of two polygons.

In the beam-triangle intersection, the part of the triangle behind the virtual aperture plane of the beam is first clipped and the remaining part is projected onto the virtual aperture plane. There are three simple cases that can be determined by checking the relative position between the beam and the projected triangle on the virtual aperture plane. When the beam is on the outside of one edge of the projected triangle or the projected triangle is on the outside of one edge of the beam, as shown in Fig. 2(a) and (b), the beam does not intersect the triangle and the intersection is terminated. When the beam is on the inside of all edges of the projected triangle, as shown in Fig. 2(c), the beam is fully inside the projected triangle. In this case, if the beam is the miss beam, we simply mark the beam as the hit beam and record the intersection information, i.e., the intersected triangle and the distance from each corner of the beam to the intersected triangle. Otherwise, if the beam is a hit beam, we need to judge the order of the previous intersected triangle and the current intersected triangle from the direction of the beam, and the nearest triangle is recorded in the beam. However, when two triangles pierce each other, that is neither of the triangles is wholly closer to the virtual aperture plane, and the beam should be split based on the intersection line of the two triangles. Furthermore, there is another simple case for the hit beam. If the current triangle is wholly behind the intersected triangle of the hit beam from the beam's direction, i.e., the distances from corners of the beam to the triangle are all further than its corresponding distances to the intersected triangle, the beam is blocked by the intersected triangle and does not intersect the triangle. In this case, there is no need to perform the intersection between the hit beam and the triangle.

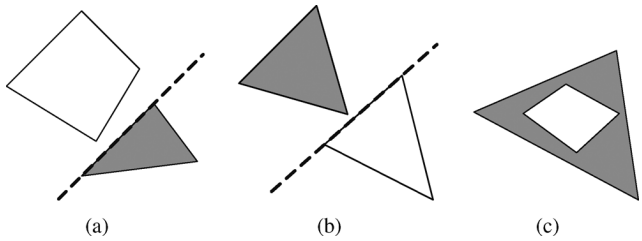

Fig. 2. The illustration of three simple cases in the beam-triangle intersection. The gray polygon is the projected triangle. (a) The beam is on one side of the projected triangle. (b) The projected triangle is on one side of the beam, which is a triangular prism. (c) The beam is fully inside the projected triangle.

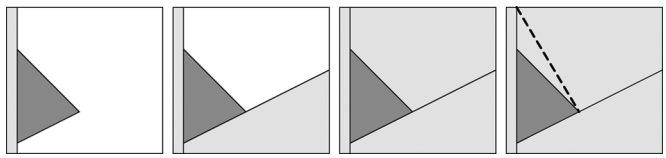

Fig. 3. The illustration of the beam-triangle intersection. The dark gray triangle is the projected triangle. The beam is iteratively split by each edge. The light gray polygon in each step is the miss beam, and the final miss beam requires an additional splitting indicated by the dashed line. The initial beam is split into one hit beam and four miss beams.

When the relationship between the beam and the triangle does not belong to the above simple cases, the beam and the projected triangle overlap on the virtual aperture plane. The classical Sutherlan-Hodgman polygon clipping algorithm [20] can be used to compute the intersection of the beam and the triangle. This algorithm deals with one triangle edge at one time. The current edge splits the beam into two parts: the miss beam and the hybrid beam, and the remaining edges only need to deal with the hybrid beam. In some situations, the miss beam may not be generated during the triangle edge's splitting. Finally, the original beam is split into several beams, at least one hit beam and one miss beam. It is worthwhile to point out that hit/miss here is relative to the current triangle. The generated miss beams have the same intersection information with the original beam, while the generated hit beams should update the intersection information. The update procedure is the same as the case that the beam is fully inside the projected triangle. Fig. 3 illustrates the splitting procedure of the beam-triangle intersection. When the corner number of the generated beam is larger than four, an additional splitting is required to ensure that the corner number is less than or equal to four for the consistency of beams.

The beam-target intersection performs the beam-triangle intersection for all triangles iteratively. In this process, old beams

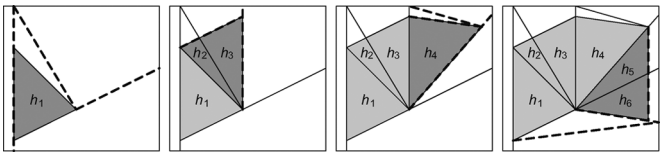

Fig. 4. The illustration of the initial beam intersection with the right-angle dihedral corner modeled by four triangles. The initial beam is iteratively split by each triangle from left to right. The dashed lines in each step are splitting lines based on the projected edges of the current processed triangle (the dark gray color), and the previous generated hit beams are indicated by the light gray color. The intersection process of the initial beam and the first triangle is illustrated in Fig. 3, and the hit beam h_1 is generated. The second triangle splits two miss beams and generates two hit beams h_2 and h_3 and three miss beams, while the third triangle splits one miss beam and generates one hit beam h_4 and three miss beams. The final triangle splits two miss beams and generates two hit beams h_5 and h_6 and four miss beams.

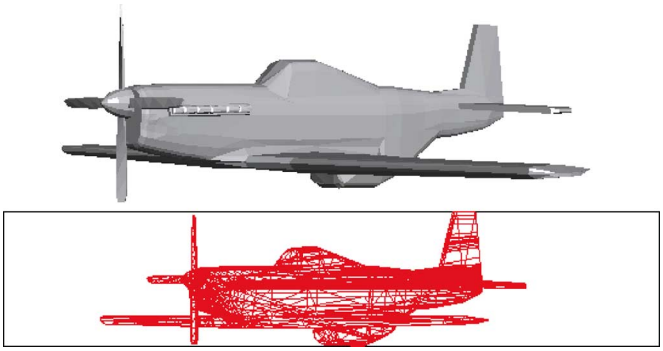

Fig. 5. The primary hit beams on the virtual aperture of the airplane from the incident direction.

are split and new beams are generated gradually, and all current beams including hit beams and miss beams continue the beam-triangle intersection for the remaining triangles. Finally, the virtual aperture is adaptively divided into irregular areas, and a group of primary beams with these irregular cross-sections are generated. Among these beams, miss beams have no intersection with the target, and each hit beam intersects only one triangle of the target. As the triangles' edges of the target are all located at the splitting lines of the virtual aperture, adaptive aperture partition generates convergent beams and solves the ray-tube divergence problem. The initial beam is the root beam, and all hit beams form the first level of the beam tree. Fig. 4 illustrates the intersection process of the initial beam with the right-angle dihedral corner, and the initial beam is divided into six primary hit beams. Fig. 5 shows primary hit beams on the virtual aperture divided based on the visible triangles of the airplane from the incident direction, and each hit beam intersects only one visible triangle of the airplane.

For multiple reflections, we take the intersected area on the hit triangle and the reflected direction of the hit beam as the virtual aperture and the propagation direction of the reflected beam. The reflected beams continue to perform the beam-target intersection, and the only difference is that all generated miss beams are hit-exit beams and should become sibling nodes of the current reflected beams. As the virtual aperture of the reflected beam is adaptively divided according to the geometry of the target, it also eliminates the divergence of the reflected beam.

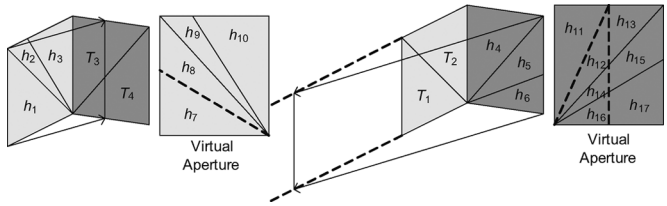

Fig. 6. The illustration of the primary reflected beam intersection with the right-angle dihedral corner. The left (right) three reflected beams and the partitions on their virtual apertures are shown in light (dark) gray color on the left (right), and the dashed lines on the virtual aperture are splitting lines. The primary hit beam h_1 intersects triangles T_3 and T_4, and is split into two secondary hit beams h_7 and h_8, while the primary hit beams h_2 (h_3) only intersects the triangle T_3, and generates one secondary hit beam h_9 (h_{10}). The primary hit beam h_4 intersects the triangle T_2 and generates one secondary hit beam h_{11}, and it also intersects the triangle T_1 and generates one secondary hit beam h_{12} and one miss beam h_{13}. The primary hit beam h_5 (h_6) only intersects the triangle T_1, and generates one secondary hit beam h_{14} (h_{16}) and one miss beam h_{15} (h_{17}).

The reflected beam is recursively traced until the beam exits the target or the number of intersections is larger than the maximum order of the reflection. Fig. 6 illustrates the reflected beams and the partition on their virtual apertures of the right-angle dihedral corner, and the reflected beams are constructed based on the primary hit beams in Fig. 4.

Finally, all hit beams constitute a beam tree. The root (zero level) of the beam tree is the initial beam without any splitting, and the nodes in the n level of the beam tree are the nth-reflection hit beams. As the virtual aperture of each level is adaptively divided based on the geometry of the target, new beams are dynamically generated to avoid the divergence problem. Fig. 7 depicts a beam tree generated from the right-angle dihedral corner. The leaf nodes are the exit beams, and electromagnetic computing is only performed on these beams.

B. Kd-Tree Beam Traversal

The beam-target intersection discussed above adaptively splits the virtual aperture based on the geometry of the target during the recursive beam tracing, and it eliminates the divergence problem in a unified manner. However, all triangles are required to be projected on the virtual aperture of each beam to perform the beam-triangle intersection. It is, therefore, very time-consuming when the geometry of the target is complex. Actually, this efficiency problem of the beam-target intersection is analogous to that of the ray-target intersection in the SBR method. Various acceleration techniques have been proposed to improve the efficiency of the ray-target intersection, such as the octree [16] and kd-tree [18]. In this section, we introduce the kd-tree to accelerate the beam-target intersection. This basic procedure is based on the traversal algorithm [7], which is used to calculate the exact soft shadows for point lighting. The essential difference is that beams in the SBR method are directional beams with the same ray directions instead of beams emitting from a point source.

The kd-tree is a simplified version of the binary space partitioning tree and has been used in the SBR method [18], [19]. The octree, which is common in computational electromagnetic [21], recursively uses the middle point of the extend in each direction as the splitting position to subdivide the target space into

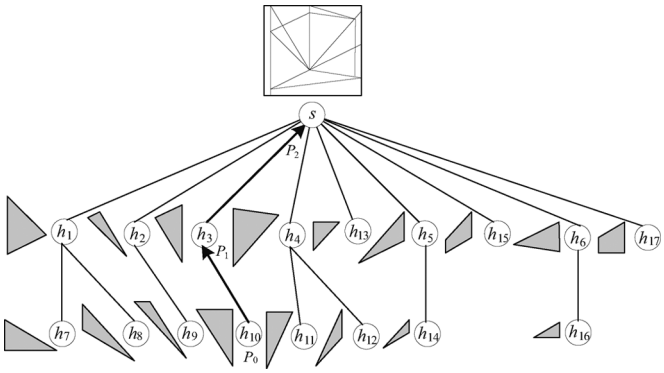

Fig. 7. The illustration of the beam tree generated from the right-angle dihedral corner. The beams in the first and second level correspond to the primary hit beams and the secondary hit beams as shown in Figs. 4 and 6, respectively. The miss beams generated in Fig. 6 are sibling nodes of the primary hit beams in Fig. 4. A representative propagation path from the leaf beam h_{10} to the initial beam s ($P_0 \rightarrow P_1 \rightarrow P_2$) is indicated in a bold line.

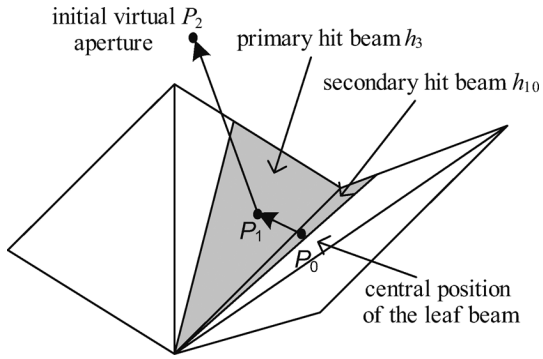

Fig. 8. The illustration of backward ray tracing. The beam tree of the right-angle dihedral corner is displayed in Fig. 7. The representative propagation path of the leaf beam h_{10} is $P_2 \rightarrow P_1 \rightarrow P_0$.

eight equal sub-spaces. The kd-tree takes into account the triangle distribution in the target space to search for the optimal splitting axis and position based on the ray-tracing cost estimation model, and recursively employs the optimal axis-perpendicular plane to divide the target space into two uneven axis-aligned sub-spaces. The ray traversal algorithm starts at the root node of the kd-tree and searches for the nearest intersected triangle of the ray in the target, and most rays could find the intersection in the first leaf nodes visited [17]. The construction procedure of the kd-tree has been introduced in [18], and Pharr and Humphreys' book [22] gives the detailed description on the ray traversal algorithm in the kd-tree.

The beam traversal procedure is based on the ray traversal procedure, as a beam can be taken as three/four corner rays in the kd-tree traversal. However, we also need to take into account other rays in the beam, and there are several differences between the two procedures: (1) the choice of the child node to traverse, (2) the maintenance of the stack, (3) the beam-triangle intersection in the leaf node, (4) the termination condition of the traversal. The beam-triangle intersection in the leaf node has been introduced in the Section III.A, and the Appendix provides detailed descriptions about other three differences and pseudocode for the interested reader.

Each beam is recursively traced based on the proposed kd-tree traversal algorithm. The use of the kd-tree can significantly accelerate the beam-target intersection, as it eliminates a large number of unnecessary beam-triangle intersections. In addition, the virtual aperture of the beam is split only by visible triangles, not all triangles, and this reduces the final number of hit beams.

IV. Beam Tree Based Electromagnetic Computing

The beam-target intersection adaptively splits the virtual aperture of each beam, and finally it generates a beam tree. A leaf beam is a group of rays with the same propagation path, and a beam tree contains all possible propagation paths. The propagation path here refers to all rays of the leaf beam are reflected by the same sequence of triangles of the target.

The scattered field of the target can be obtained by evaluating the scattered fields of leaf beams only. We calculate the central position P_0 of the leaf beam's virtual aperture, and employ backward ray tracing to produce a representative ray path for each leaf beam. More specifically, the origin and the direction of the backward ray are the central position P_0 and the inverse propagation direction of its parent beam, respectively. The backward ray is tested for intersection with the virtual aperture of its parent beam, and it obtains the intersection position P_1. The origin and the direction of the backward ray are replaced with the intersection position P_1 and the inverse propagation direction of the corresponding parent beam, respectively. The backward ray is recursively traced until it hits the virtual aperture of the root beam and produces the final intersection position P_n. Thus, the representative ray path of the leaf beam is from P_n to P_0: $P_n \rightarrow P_{n-1} \rightarrow \cdots \rightarrow P_1 \rightarrow P_0$. Fig. 8 displays an example of backward ray tracing in the right-angle dihedral corner, and its beam tree is illustrated in Fig. 7.

Geometric optics is applied to each intersection of the representative ray path to evaluate the exit field of the leaf beam. The reflected field is calculated based on the field before the intersection and the geometric information of the intersected triangle as follows:

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

where $E_{\parallel}^{i} = \hat{\mathbf{e}}_{\parallel} \cdot \mathbf{E}^{\mathbf{i}}$, $E_{\perp}^{i} = \hat{\mathbf{e}}_{\perp} \cdot \mathbf{E}^{\mathbf{i}}$, $\hat{\mathbf{e}}_{\perp} = \hat{\mathbf{k}}^{\mathbf{i}} \times \hat{\mathbf{n}} / |\hat{\mathbf{k}}^{\mathbf{i}} \times \hat{\mathbf{n}}|$, $\hat{\mathbf{e}}_{\parallel} = \hat{\mathbf{k}}^{\mathbf{i}} \times \hat{\mathbf{e}}_{\perp} / |\hat{\mathbf{k}}^{\mathbf{i}} \times \hat{\mathbf{e}}_{\perp}|$, and $\hat{\mathbf{e}}_{\parallel}^{\mathbf{r}} = \hat{\mathbf{k}}^{\mathbf{r}} \times \hat{\mathbf{e}}_{\perp} / |\hat{\mathbf{k}}^{\mathbf{r}} \times \hat{\mathbf{e}}_{\perp}|$. The vector $\hat{\mathbf{k}}^{\mathbf{i}}$ is the propagation direction before the intersection, $\hat{\mathbf{k}}^{\mathbf{r}}$ is the propagation direction after the intersection, and $\hat{\mathbf{n}}$ is the normal of the intersection. The incident field is $\mathbf{E}^{\mathbf{i}}$ and the reflected field is $\mathbf{E}^{\mathbf{r}} = \hat{\mathbf{e}}_{\parallel}^{\mathbf{r}} \cdot E_{\parallel}^{r} + \hat{\mathbf{e}}_{\perp} \cdot E_{\perp}^{r}$. The detailed formulas about the reflection coefficients are explained in [1], [3].

Since the leaf beam is reflected by a group of triangles, the exit field on the leaf beam has the same amplitude and a linear phase variation with the exit field of the representative ray. The scattered field of the leaf beam can be approximated by the PO integral as follows:

$$\mathbf{E}(r, \theta, \phi) \approx \frac{e^{-jkr}}{r} (\hat{\boldsymbol{\theta}} E_{\theta} + \hat{\boldsymbol{\phi}} E_{\phi}) \tag{2}$$

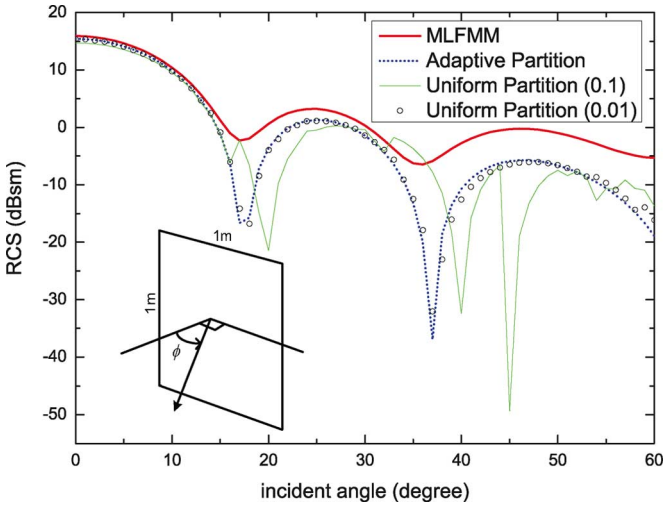

Fig. 9. The VV-polarization comparison of uniform aperture partition, adaptive aperture partition and MLFMM results for a simple 1 m × 1 m square patch at 500 MHz.

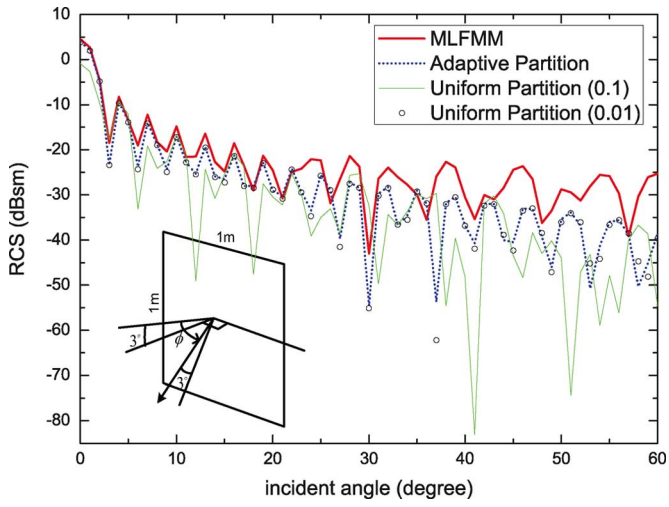

Fig. 10. The VV-polarization comparison of uniform aperture partition, adaptive aperture partition and MLFMM results for a simple 1 m × 1 m square patch at 3 GHz.

where (θ, ϕ) is the observation direction. The (E_θ, E_ϕ) can be expressed as the exit field $(\mathbf{E}, \mathbf{H})$ of the polygon S

$$\begin{bmatrix} E_\theta \\ E_\phi \end{bmatrix} = \left(\frac{jk}{2\pi}\right) \iint_S e^{j\mathbf{k}\cdot\mathbf{r}'} \left\{ \begin{bmatrix} -\hat{\boldsymbol{\phi}} \\ \hat{\boldsymbol{\theta}} \end{bmatrix} \times \mathbf{E}(r') f_e + Z_0 \begin{bmatrix} \hat{\boldsymbol{\theta}} \\ \hat{\boldsymbol{\phi}} \end{bmatrix} \times \mathbf{H}(r') f_h \right\} \cdot \hat{\mathbf{n}} dx' dy'. \quad (3)$$

Generally, the coefficients f_e and f_h in the EH formulation (0.5) provide a better result [2]. The PO integral on the planar aperture can be approximately in a more computable form [2], and the shape function can be solved through the 2D Fourier transform [23]. The scattered field of the target is generated by summing all leaf beams' scattered fields.

V. Results and Discussion

Several numerical experiments were performed to verify the accuracy and efficiency of the proposed adaptive aperture partition. The original partition of the SBR method is referred to as uniform aperture partition in this section. In our implementation, the identified divergent ray tubes are simply discarded in uniform aperture partition. These experiments were tested on an Intel Core 2 Quad Q9550 (2.83 GHz) processor with an NVIDIA GeForce 285 GTX (CUDA Toolkit 3.0), and at most fifth-order reflection is considered. The target is described by triangles and meshed according to the geometric error instead of the wavelength. Thus, it is desirable that the flat part is modeled by large triangles, as this reduces the beam-triangle intersections and avoids many unnecessary beam splittings.

A simple 1 m × 1 m square patch is used to analyze the influence of the ray-tube divergence problem on the accuracy of the SBR method. Fig. 9 shows the VV-polarization RCS result, which are predicted using an angular resolution of 1° at 500 MHz frequency with $\theta = 90°$. Adaptive aperture partition actually generates two large hit beams covering the whole patch, and the result is equal to the PO integral of the total patch. The result of uniform aperture partition at 0.1λ is largely different from the one of adaptive aperture partition. The ray-tube divergence problem severely affects the accuracy in this case, since ray tubes that intersect the boundary of the patch are not included in the calculation of the scattered field and the ray-tube size (0.1λ) is considerable large compared to the target. The result of uniform aperture partition at 0.01λ is also shown in Fig. 9. In this case, the relative error due to the ray-tube divergence problem is greatly reduced, and the result tends toward the one of adaptive aperture partition. The result of adaptive aperture partition which eliminates the ray-tube divergence problem is more accurate than the one of uniform aperture partition in the SBR method. In other words, adaptive aperture partition can generate the convergent solution of the SBR method when the ray-tube size tends toward infinitesimal. The result of the multilevel fast multipole method (MLFMM) is used as a comparison to verify the result. As PO is mainly valid in the nearby direction of specular reflection and it is also not accurate enough in the low frequency, there are some deviations between the results of the SBR method and MLFMM. However, the result of adaptive aperture partition matches the MLFMM result and is more accurate than the one of uniform aperture partition.

With the increasing frequency, the SBR method is highly effective in predicting the scattered field of arbitrarily shaped targets. However, the ray-tube divergence problem still affects the accuracy of the SBR method. Fig. 10 shows the VV-polarization RCS result for the same square patch, which are predicted using an angular resolution of 1° at 3 GHz frequency with $\theta = 87°$. As can be observed clearly from Fig. 10, there are obvious differences between the results of uniform aperture partition at 0.1λ and adaptive aperture partition, and the result of uniform aperture partition at 0.01λ tends to the one of adaptive aperture partition. Adaptive aperture partition generates the optimal result of the SBR method and its result is more similar to the MLFMM result. In addition, the result of uniform aperture partition depends on the boundary of the virtual aperture and the partition criterion, and there are differences among results under different configurations of the virtual aperture. In contrast to

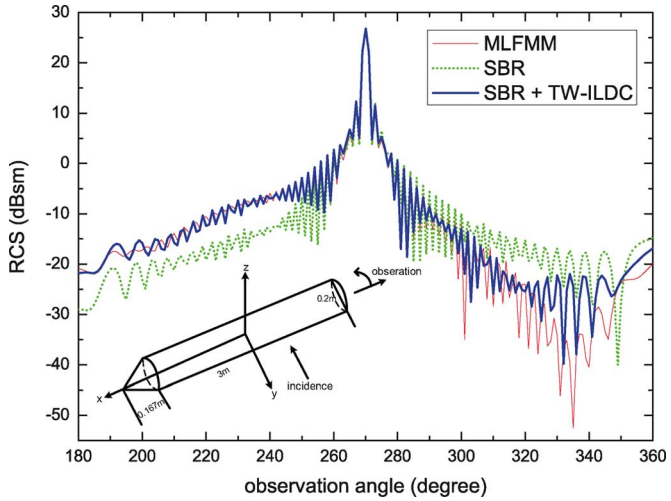

Fig. 11. The VV-polarization comparison of adaptive aperture partition, adaptive aperture partition + TW-ILDC and MLFMM results in the bistatic RCS calculation for the pencil at 3 GHz.

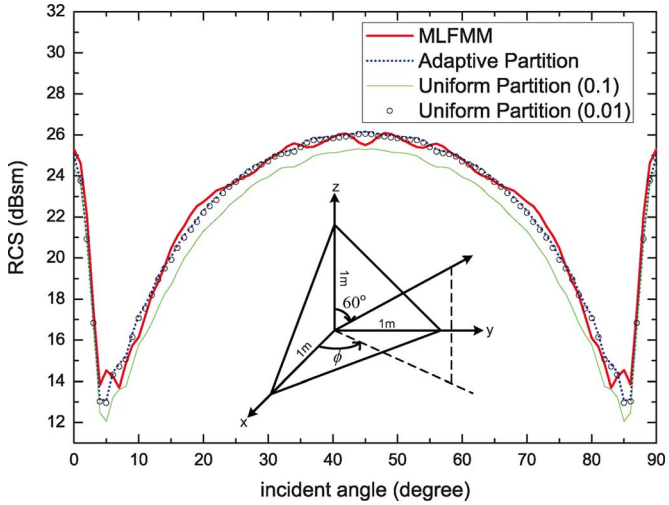

Fig. 12. The HH-polarization comparison of uniform aperture partition, adaptive aperture partition and MLFMM results in the monostatic RCS calculation for the trihedral corner reflector at 3 GHz.

uniform aperture partition, the result of adaptive aperture partition does not rely on the boundary of the virtual aperture and the order of partition. Thus, adaptive aperture partition is more stable than uniform aperture partition. The agreement between adaptive aperture partition and MLFMM is less good for angles higher than 35°, as the observation direction in these angles deviates from the specular reflection direction, and PO could not evaluate the scattered field accurately in this case.

The pencil illustrated in Fig. 11 is used to verify the accuracy of adaptive aperture partition in the SBR method. It was first employed by Hastriter [24] to verify the Fast Illinois Solver Code (FISC), and then it was also used to verify the effectiveness of truncated-wedge incremental-length diffraction coefficients (TW-ILDC) [25], [26]. The incident direction is at $\theta = 90°$ and $\phi = -90°$, and the observation directions are from $\phi = 180°$ to $\phi = 360°$ on the $\theta = 90°$ plane using an angular resolution of 1°. These calculation parameters are the same as [26]. Fig. 11 illustrates the bistatic VV-polarization RCS comparison of adaptive aperture partition and MLFMM results at 3 GHz. When specular scattering is not the dominant mechanism, the edge-diffraction effect could not be ignored. The visible edges can be identified by checking the boundaries of primary hit beams, as the projected triangles' edges are the splitting lines of beams. Then, TW-ILDC is applied to these visible edges to evaluate the edge diffracted field, and the detailed formulas of TW-ILDC are described in [25]. It is clear that the SBR + TW-ILDC result matches well the MLFMM result, and the SBR + TW-ILDC result is nearly the same as the one in [26]. The deviation in observation angles higher than 300° may be due to the pencil-tip diffraction, which is not included in our current implementation.

The trihedral corner reflector, which is a typical benchmark target, is used to verify the high frequency multiple-bounce scattering [2]. The geometry of trihedral corner reflector is depicted in Fig. 12, three right-angled triangles with the side length 1 m. The 91 equal-spaced incident directions are from $\phi = 0°$ to $\phi = 90°$ on the $\theta = 60°$ plane. The monostatic HH-polarization RCS comparison of uniform aperture partition, adaptive aperture partition, and MLFMM results at 3 GHz is illustrated in Fig. 12. The result of uniform aperture partition at 0.1λ is smaller than the one of adaptive aperture partition due to the ray-tube divergence problem. A good agreement between the adaptive aperture partition and MLFMM results further verifies the accuracy of adaptive aperture partition in the SBR method.

The impact of the ray-tube divergence problem is more obvious for electrically large and complex targets. We calculate the scattered field of the airplane as shown in Fig. 5 to show such influence on accuracy. The size of the airplane is approximately 14 m × 17 m × 4.5 m, and the structure of the airplane is much more complex. The 181 equal-spaced incident directions are from $\phi = 0°$ to $\phi = 180°$ on the $\theta = 90°$ plane. Fig. 13 shows the RCS comparison of adaptive aperture partition and uniform aperture partition results at 3 GHz. It can be seen obviously that the result of uniform aperture partition is slightly smaller than the one of adaptive aperture partition due to the discarding of divergent ray tubes. There is no comparison with the MLFMM result, as the MLFMM becomes unusable for the airplane due to the limited computational resources.

Table I shows the total computational times of all incident angles using CPU uniform aperture partition, GPU uniform aperture partition, and the proposed adaptive aperture partition for experimented targets. As demonstrated in [19], GPU uniform aperture partition is at least 30 times faster than CPU uniform aperture partition for most cases. Especially, it fully exploits the potential of GPU at 0.01λ, achieving an acceleration ratio about 100 for the trihedral corner reflector and airplane. As the trihedral corner reflector has a very simple geometric shape, adaptive aperture partition only needs 0.078 seconds for all incident angles, while the computational times of CPU uniform aperture partition at 0.1λ and 0.01λ are 18.03 and 1974.76 seconds, respectively. Adaptive aperture partition is even faster than GPU uniform aperture partition for this target, about 2 and 60 times faster at 0.1λ and 0.01λ, respectively. Thus, adaptive aperture partition is very well suited to the target with large flat regions, such as the patch and trihedral corner reflector.

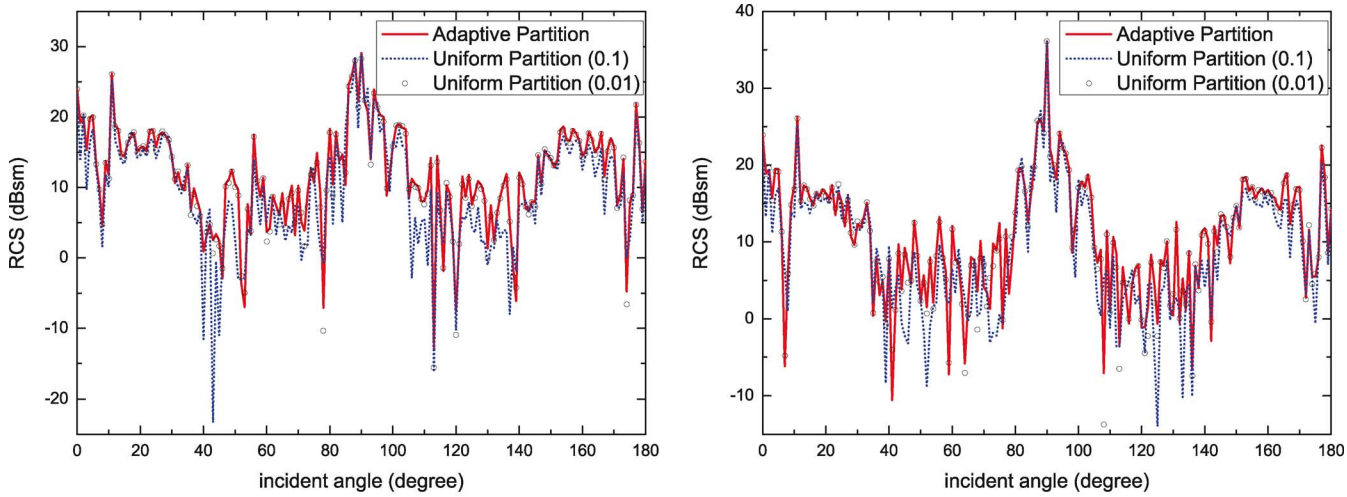

Fig. 13. The comparison of uniform aperture partition and adaptive aperture partition results in the monostatic RCS calculation for the airplane as shown in Fig. 5 at 3 GHz. (a) VV-polarization result, (b) HH-polarization result.

TABLE I
THE COMPUTATIONAL TIMES OF THE CPU UNIFORM APERTURE PARTITION, GPU UNIFORM APERTURE PARTITION [19], AND ADAPTIVE APERTURE PARTITION FOR THE RESULTS IN FIGS. 9–14 (SEC)

Targets	CPU Uniform		GPU Uniform		Adaptive
	0.1λ	0.01λ	0.1λ	0.01λ	
Patch (500M)	0.160	13.198	0.065	0.084	0.019
Patch (3G)	4.754	488.192	0.076	0.715	0.022
Pencil (3G)	2.856	969.106	1.63	7.65	818.31
Trihedral (3G)	18.03	1974.76	0.18	4.66	0.078
Airplane (3G)	677.26	79033.69	16.97	487.77	4277.42
Pencil (1T)	>1 day	>1 day	3384.81	>1 day	822.40

In computer graphics, beam tracing is much slower than ray tracing for models of complex structures, especially when the pixel size on the sampling plane is larger than the average visible triangle size. In the SBR method, adaptive aperture partition also has such a problem. The beam number is proportional to the number of visible triangles. In our experiments, the average primary hit beam number/visible triangle number is 292609/33874, 5/3, and 16251/1128 for the targets in Figs. 11, 12, and 13, respectively. This ratio reaches up to 14.4 for the airplane. The reason is that as visible triangles split the beam iteratively, the area on the virtual aperture corresponding to one visible triangle may be already split into several beams before encountering this visible triangle. This also explains why adaptive aperture partition may be slower than uniform aperture partition in some situations, such as the pencil and the airplane. As shown in Table I, adaptive aperture partition is much slower than CPU uniform aperture partition at 0.1λ, as it spends additional computational burden on the generation of a high number of beams and the size of many beams would be much smaller than ray tubes in uniform aperture partition. However, the accuracy of adaptive aperture partition is greatly improved compared to uniform aperture partition at 0.1λ. Although the result of uniform aperture partition at 0.01λ is very similar to the one of adaptive aperture partition, the computational time is terribly long due to a large number of ray tubes. For example, the total computational time of CPU uniform aperture partition at 0.01λ is about 100 times slower than uniform aperture partition at 0.1λ for the airplane, and adaptive aperture partition is faster than CPU uniform aperture partition at 0.01λ for the pencil and airplane. Although the performance of adaptive aperture partition is much worse than GPU uniform aperture partition, adaptive aperture partition can also explore the GPU power for acceleration, which would be our future work.

Another feature of adaptive aperture partition is that the partition of the virtual aperture is insensitive to the incident frequency, and it is only related to the geometry of the target. This is particularly useful for the high-resolution range and inverse synthetic aperture radar (ISAR) applications, since we only need to generate one beam tree for all frequencies in the bandwidth.

In the terahertz (THz) band, the electrical size of the target is extremely large, and the number of ray tubes in uniform aperture partition is also significantly increased. Thus, the computational time is seriously affected by such a large number of ray tubes [27]. Adaptive aperture partition is very suited to the THz band, as it is no concern of the frequency. For instance, Fig. 14 shows the RCS result of the pencil at 1 THz, and other computational parameters are the same to the one in Fig. 11. As can be seen from Table I, the computational time of adaptive aperture partition is almost the same for all frequencies, 822.40 seconds, while the computational time of GPU uniform aperture partition at 0.1λ is 3384.81 seconds, about 4 times slower, and its computational time at 0.01λ is extremely time-consuming, more than one day. Compared to the result in the GHz as shown in Fig. 11, the maximum/minimum RCS value is increased/decreased in the THz.

VI. CONCLUSION

This paper presents an adaptive aperture partition algorithm to solve the ray-tube divergence problem, and also proposes the kd-tree to accelerate the beam-target intersection. Adaptive aperture partition is more stable than uniform aperture partition in the original SBR method and yields the optimal result of the

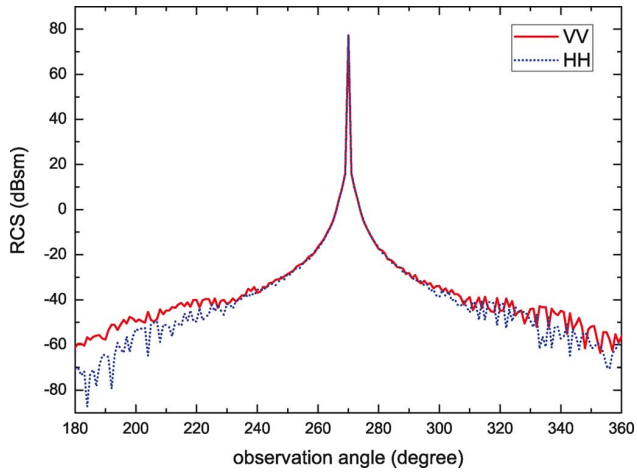

Fig. 14. The bistatic RCS result of the pencil at 1 THz.

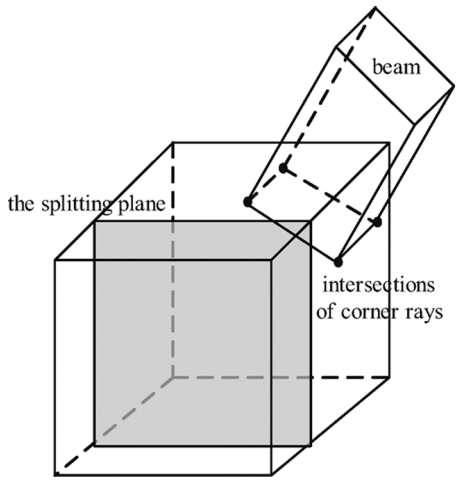

Fig. 15. The illustration of the beam traversal. All four corner rays of the beam need to visit the far node (the back node) only. However, the beam hits the near node and should traverse the near node firstly.

SBR method. This is because adaptive aperture partition takes into account the geometry of the target and the generated hit beams are convergent. In contrast, the result of uniform aperture partition is sensitive to the boundary of the virtual aperture and the partition criterion to some extent, and it is also influenced by the divergent ray tubes for complex targets. Another feature of adaptive aperture partition is that the aperture partition depends only on the geometry of the target, and it is insensitive to the incident frequency. Therefore, adaptive aperture partition can significantly accelerate the high-resolution range and inverse synthetic aperture radar (ISAR) applications as well as the RCS prediction in the THz band. Numerical results demonstrate the accuracy and effectiveness of the proposed algorithm. Furthermore, the beam-target intersection accelerated by the kd-tree can be adapted to other electromagnetic applications, such as the radio propagation prediction [10]–[12].

APPENDIX
DIFFERENCES BETWEEN RAY TRAVERSAL AND BEAM TRAVERSAL IN THE KD-TREE

The other three differences between ray traversal and beam traversal in the kd-tree are described in detail as follows:

a) The Choice of the Child Node to Traverse: When a beam encounters an interior node, we first need to choose which child node to traverse. The near and far child nodes are decided by the beam's corners with respect to the splitting plane. If the corners straddle the splitting plane, the beam' propagation direction is taken into account to determine the near and far nodes. It is not accurate enough to simply use the beam's corner rays to determine the traversal order of all rays in the beam [7], [28]. As illustrated in Fig. 15, all four corner rays of the beam need to traverse the far node only. However, the rays in the beam actually hit the near node, and the near node should be traversed first. This problem can be solved by maintaining three $(t_{\min}, t_{\max})$ ranges, i.e., one range for each axis, and each range defines the part of the ray within two bounding planes perpendicular to its corresponding axis. If distances from the corners to the splitting plane are all further than either of the other axes' maximum distances $(t_{\text{split}} > t_{\max}[\text{axis}])$, or all corner rays face away from the far node $(t_{\text{split}} < 0)$ only the near node needs to be traversed. If distances are all less than either of the other axes' minimum distances $(t_{\text{split}} < t_{\min}[\text{axis}])$, only the far node needs to be traversed. Otherwise, both nodes need to be traversed.

b) The Maintenance of the Stack: The ray traversal employs a node stack as a priority queue of nodes left to visit according to how close to the origin of the ray. The ray is not changed during the ray traversal, but the beam would be split and gradually generate new beams during the beam traversal. As a result, besides the node stack, the beam traversal requires an additional beam stack. The node stack preserves the to-be-visited tree node and its corresponding beam, and the beam stack maintains the new generated to-be-processed beam.

c) The Termination Condition of the Traversal: In the ray traversal, when the ray's distance to a triangle is less than the ray's maximum distance to the node, the intersection is considered as the nearest intersection along the ray and the traversal terminates. Likewise, when the corner rays' distances to the triangle are all less than the corner rays' maximum distances to the node in any dimension, the triangle can be guaranteed to the nearest triangle along the beam and the hit beam can stop its traversal. Otherwise, the hit beam still needs to traverse the kd-tree until the entry distance to the next to-be-visited node in any one axis is larger than the distances to the hit triangle. The miss beam terminates its traversal after passing through the target space, i.e., there is no element left in the node stack.

After we discussed the differences between the ray traversal procedure and the beam traversal procedure, a kd-tree traversal algorithm for beam tracing is shown in Algorithm 1. It should be noted that the comparison between t_{split} and $t_{\min}$ as well as $t_{\max}$ corresponds to the array comparison of all corner rays. Each node in the node stack should be traversed by all new generated beams after the node is pushed into *nodeStack*. The function *beamTriIntersect* deals with the intersection of the beam with one triangle in the leaf node, and keeps the original beam or new generated beams if split into *stack*. All beams in the *stack*

should be tested for intersection with the remaining triangles to find the nearest intersected triangle. Finally, the input beam is split into a group of beams, and these beams are children beams of the input beam.

Algorithm 1 Kd-Tree Traversal Algorithm for Beam Tracing

$mod \leftarrow \{1,2,0,1\}$
$node \leftarrow root$
$beam = (corners, direction)$
while $node \neq nil$ and $beam \neq nil$ **do**
  $(t_{max}, t_{min}) \leftarrow intersect(BoundingBox(node), beam)$

  // process interior nodes
  while $\neg isLeaf(node)$ **do**
    $axis \leftarrow node.splitAxis$
    $t_{split} \leftarrow calcSplitDist(node, beam)$
    $(nearNode, farNode) \leftarrow getNearFarNode(node, beam)$
    if $t_{split} < 0$ or $t_{split} > t_{\max}[mod[axis]]$ or $t_{split} > t_{\max}[mod[axis+1]]$ **then**
      $node \leftarrow nearNode$
    else if $t_{split} < t_{\min}[mod[axis]]$ or $t_{split} < t_{\min}[mod[axis+1]]$ **then**
      $node \leftarrow farNode$
    else
      $nodeStack.push(farNode, beamStack.size())$
      $node \leftarrow nearNode$
      $t_{max}[axis] \leftarrow t_{split}$
    end if
  end while

  // intersection test with the triangles in the leaf node
  stack.push(*beam*)
  for *triangle* **in** *node.triangles* **do**
    for *beam* **in** *stack* **do**
      beamTriIntersect(*beam*, *triangle*, *stack*)
    end for
  end for
  beamStack.push(*stack*)

  // get the next beam and node from the stack
  (*beam*, *node*) ← *getBeamNode*(*nodeStack*, *beamStack*)
end while
return beam

ACKNOWLEDGMENT

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

REFERENCES

[1] H. Ling, R. C. Chow, and S. W. Lee, "Shooting and bouncing rays: Calculating the RCS of an arbitrarily shaped cavity," *IEEE Trans. Antennas Propag.*, vol. 37, no. 2, pp. 194–205, 1989.

[2] J. Baldauf, 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, pp. 1345–1351, 1991.

[3] C. A. Balanis, *Advanced Engineering Electromagnetics*. New York: Wiley, 1989.

[4] M. L. Hastriter and W. C. Chew, "Comparing Xpatch, FISC, and ScaleME using a cone-cylinder," in *Proc. IEEE Antennas and Propag. Society Int. Symp.*, Monterrey, CA, Jun. 2004, vol. 2, pp. 2007–2010.

[5] P. S. Heckbert and P. Hanrahan, "Beam tracing polygonal objects," in *Proc. SIGGRAPH'84*, New York, 1984, pp. 119–127.

[6] D. Ghazonfarpour and J.-M. Hasenfratz, "A beam tracing method with precise antialiasing for polyhedral scenes," *Comput. Graph.*, vol. 22, no. 1, pp. 103–115, 1998.

[7] R. Overbeck, R. Ramamoorthi, and W. R. Mark, "A real-time beam tracer with application to exact soft shadows," in *Proc. EuroGraphics Symp. on Rendering*, Jun. 2007, pp. 85–98.

[8] T. Funkhouser, I. Carlbom, G. Elko, G. Pingali, M. Sondhi, and J. West, "A beam tracing approach to acoustic modeling for interactive virtual environments," in *Proc. SIGGRAPH'98*, New York, 1998, pp. 21–32.

[9] N. Tsingos, T. Funkhouser, A. Ngan, and I. Carlbom, "Modeling acoustics in virtual environments using the uniform theory of diffraction," in *Proc. SIGGRAPH'01*, New York, 2001, pp. 545–552.

[10] H.-W. Son and N.-H. Myung, "A deterministic ray tube method for microcellular wave propagation prediction model," *IEEE Trans. Antennas Propag.*, vol. 47, no. 8, pp. 1344–1350, 1999.

[11] P. Bernardi, R. Cicchetti, and O. Testa, "An accurate UTD model for the analysis of complex indoor radio environments in microwave WLAN systems," *IEEE Trans. Antennas Propag.*, vol. 52, no. 6, pp. 1509–1520, 2004.

[12] H. Kim and H. Lee, "Accelerated three dimensional ray tracing techniques using ray frustums for wireless propagation models," *Progress Electromagn. Res. (PIER)*, vol. 96, pp. 21–36, 2009.

[13] S. H. Suk, 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, pp. 394–397, 2001.

[14] F. Weinmann, "Ray tracing with PO/PTD for RCS modeling of large complex objects," *IEEE Trans. Antennas Propag.*, vol. 54, no. 6, pp. 1797–1806, 2006.

[15] F. Xu and Y.-Q. Jin, "Bidirectional analytic ray tracing for fast computation of composite scattering from electric-large target over a randomly rough surface," *IEEE Trans. Antennas Propag.*, vol. 57, no. 5, pp. 1495–1505, 2009.

[16] K. S. Jin, T. I. Suh, S. H. Suk, B. C. Kim, and H. T. Kim, "Fast ray tracing using a space-division algorithm for RCS prediction," *J. Electromagn. Waves Applicat.*, vol. 20, no. 1, pp. 119–126, 2006.

[17] V. Havran, "Heuristic ray shooting algorithms" Ph.D. dissertation, Univ. of Czech Technical, Prague, Nov. 2000 [Online]. Available: http://www.cgg.cvut.cz/~havran/phdthesis.html

[18] Y.-B. Tao, H. Lin, and H.-J. Bao, "Kd-tree based fast ray tracing for RCS prediction," *Progress Electromagn. Res. (PIER)*, vol. 81, pp. 329–341, 2008.

[19] Y.-B. Tao, 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, pp. 494–502, 2010.

[20] D. D. Hearn and M. P. Baker, *Computer Graphics With Open GL*, 3rd ed. Englewood Cliffs, NJ: Prentice Hall, 2003.

[21] W. C. Chew, J. M. Jin, E. Michielssen, and J. M. Song, *Fast and Efficient Algorithms in Computational Electromagnetics*. Boston, MA: Artech House, 2001.

[22] M. Pharr and G. Humphreys, *Physically Based Rendering: From Theory to Implementation*. San Fransisco, CA: Morgan Kaufmann, 2004.

[23] S.-W. Lee and R. Mittra, "Fourier transform of a polygonal shape function and its application in electromagnetics," *IEEE Trans. Antennas Propag.*, vol. 31, no. 1, pp. 99–103, 1983.

[24] M. L. Hastriter, "A study of MLFMA for large-scale scattering problems," Ph.D. dissertation, Univ. of Illinois at Urbana-Champaign, Champaign, IL, 2003.

[25] P. M. Johansen, "Uniform physical theory of diffraction equivalent edge currents for truncated wedge strips," *IEEE Trans. Antennas Propag.*, vol. 44, no. 7, pp. 989–995, 1996.

[26] J. T. Moore, A. D. Yaghjian, and R. A. Shore, "Shadow boundary and truncated wedge ILDCs in Xpatch," in *Proc. IEEE Antennas and Propag. Society Int. Symp.*, 2005, vol. 1, pp. 10–13.

[27] Z. Li, T.-J. Cui, X.-J. Zhong, Y.-B. Tao, and H. Lin, "Electromagnetic scattering characteristics of PRC targets in the terahertz regime," *IEEE Antennas Propag. Mag.*, vol. 51, no. 1, pp. 39–50, 2009.
[28] A. Reshetov, A. Soupikov, and J. Hurley, "Multi-level ray tracing algorithm," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 1176–1185, 2005.

Yubo Tao received the B.S. and Ph.D. degree in computer science and technology from Zhejiang University, Hangzhou, China, in 2003 and 2009, respectively.

He is currently a Postdoctoral Researcher in the State Key Laboratory of CAD&CG, Zhejiang University. His research interests include computational electromagnetics and data visualization.

Hai Lin received the B.Sc. and M.Sc. degrees in electrical engineering from Xidian University, Xi'an, China, in 1987 and 1990, respectively, and the Ph.D. degree in computer science from Zhejiang University, Hangzhou, China, in 2000.

Currently, he is a Professor of visual computing in the State Key Lab. of CAD&CG, Zhejiang University. He is also a Visiting Professor at the Department of Computing and Information Systems, University of Bedfordshire, U.K. His research interests include computational electromagnetic, computer graphics, scientific visualization.

Hujun Bao received the B.S. and Ph.D. degrees in applied mathematics from Zhejiang University, Hangzhou, China, in 1987 and 1993, respectively.

Currently, he is a Professor and the Director of State Key Laboratory of CAD&CG, Zhejiang University. His main research interest is computer graphics and computer vision, including realtime rendering technique, geometry computing, virtual reality and 3D reconstruction.