Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

A new parallel meshing technique integrated into the conformal FDTD method for solving complex electromagnetic problems*

Yang GUO†, Xiang-hua WANG, Jun HU†‡
(*Centre for Optical and Electromagnetic Research, State Key Lab of MOI, Zhejiang University, Hangzhou 310058, China*)
†E-mail: guoyang@coer-zju.org; hujun@zju.edu.cn

Received Apr. 14, 2014; Revision accepted June 18, 2014; Crosschecked Nov. 13, 2014

Abstract: A new efficient parallel finite-difference time-domain (FDTD) meshing algorithm, based on the ray tracing technique, is proposed in this paper. This algorithm can be applied to construct various FDTD meshes, such as regular and conformal ones. The Microsoft F# language is used for the algorithm coding, where all variables are unchangeable with its parallelization advantage being fully exploited. An improved conformal FDTD algorithm, also integrated with an improved surface current algorithm, is presented to simulate some complex 3D models, such as a sphere ball made of eight different materials, a tank, a J-10 aircraft, and an aircraft carrier with 20 aircrafts. Both efficiency and capability of the developed parallel FDTD algorithm are validated. The algorithm is applied to characterize the induced surface current distribution on an aircraft or a warship.

Key words: Finite-difference time-domain (FDTD), Meshing, Parallel, Function language, Surface current distribution
doi:10.1631/jzus.C1400135 **Document code:** A **CLC number:** TP391; O44

1 Introduction

Various electromagnetic problems (Shan *et al.*, 2013; Wang *et al.*, 2014) have been solved using the finite-difference time-domain (FDTD) method due to its strong capabilities (Taflove and Hagness, 2000). However, when it is used to simulate a structure, you always need to generate its applicable FDTD meshes. In particular for a conformal FDTD algorithm, an appropriate meshing is very important because it is directly related to its simulation accuracy. In fact, most commercial FDTD softwares, such as XFDTD (Remcom, USA) and GEMS (2Comu, USA), do have their own meshing modules, and in addition, there are some professional meshing tools (Hill, 1996; Yang and Chen, 1999; Srisukh *et al.*, 2002; Flubacher and Luebbers, 2003; Benkler *et al.*, 2008).

The real-world model, such as an aircraft or a ship, often contains millions of surface patches. Thus, its meshing process would take several hours and even much longer. Under such circumstances, an advanced parallel meshing technique will be useful for shortening the meshing time. To the best of our knowledge, although parallel FDTD simulations have been recently achieved (Yu and Mittra, 2000; Guiffaut and Mahdjoubi, 2001; Lei *et al.*, 2008), there is still a large area for the improvement of their efficiencies for quickly solving complex electromagnetic problems in the presence (absence) of an intentional electromagnetic interference (IEMI) (Hadi and Mahmoud, 2007; Vaccari *et al.*, 2011; Xiong *et al.*, 2012).

Note that it is not convenient to use the traditional coding C-language to develop a parallel meshing algorithm for FDTD simulation. A function language would be a better choice, as its advantage for meshing parallelization can be comprehensively exploited. During the implementation of a parallel FDTD algorithm, its post-process is also very

‡ Corresponding author

* Project supported in part by the National Natural Science Foundation of China (No. 60831002) and Zhejiang Provincial Natural Science Foundation of China (No. LZF010001)

ORCID: Yang GUO, http://orcid.org/0000-0002-5681-0606 important. However, the stair-case error of the Yee's FDTD cell often degrades its simulation accuracy while characterizing the surface current distribution of an arbitrary perfect electric conductor (PEC) structure. Under such circumstances, a finite-element time-domain (FETD) method based on unstructured meshes should be an alternative, and it can remove the stair-case error in an FDTD simulation (Kim and Teixeira, 2011). To relieve such a problem, an appropriate approximation for FDTD meshing can be employed, and some studies on this topic can be found (Juntunen and Tsiboukis, 2000; Hsu *et al.*, 2009; Wang and Yin, 2013). However, for a complex PEC geometry, the accuracy of an FDTD simulation for its surface current distribution is still lower in comparison with the finite element and integral equation methods.

In this paper, an efficient parallel FDTD meshing technique is proposed, and it is further integrated into the conformal FDTD simulation for handling electrically large 3D structures.

2 Meshing methodology

Traditional meshing techniques usually use searching-based parity count methods (Hill, 1996); i.e., one searching origin point is selected for an arbitrary model, and then a line between the origin point and the center of an FDTD cell is drawn. The FDTD grid must be inside the model when the interception point number is odd; otherwise, the grid should be outside the model. With this approach, some problems will be caused by singularity and boundary (Hill, 1996); in addition, the searching-based parity count algorithm is not parallelizable in nature. Here, we present a flexible parallel ray tracing technique for FDTD meshing which includes both high efficiency and high accuracy.

2.1 Ray tracing technique

Most 3D models, which are built using professional modeling software, are usually described by their surfaces, and they always consist of a huge number of triangle pieces. Therefore, it is important to snip these triangles into the FDTD rectangular meshes. To do this, we need to first extract all interception points between the triangles and rectangles. Fig. 1 shows the ray tracing technique implemented for treating an almond-shaped model, and when using a ray to penetrate it, we usually obtain two interception points.

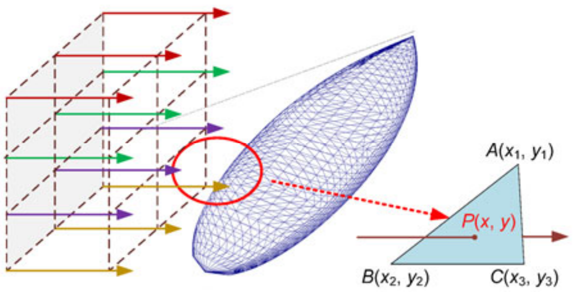

Fig. 1 The scheme of ray tracing technique

In Fig. 1, the coordinates of triangle *ABC* are denoted by $\boldsymbol{A}=(x_1, y_1)$, $\boldsymbol{B}=(x_2, y_2)$, and $\boldsymbol{C}=(x_3, y_3)$ in the *ABC* local plane. Then an arbitrary location in this local plane can be represented using the method of vector superposition, i.e.,

$$\boldsymbol{P}(x, y)=u\boldsymbol{A}+v\boldsymbol{B}+w\boldsymbol{C}, \tag{1}$$

where

$$\begin{cases} u=\dfrac{x_2y_3+x_3y+xy_2-x_2y-xy_3-x_3y_2}{x_2y_3+x_3y_1+x_1y_2-x_2y_1-x_1y_3-x_3y_2}, \\ v=\dfrac{xy_3+x_3y_1+x_1y-xy_1-x_1y_3-x_3y}{x_2y_3+x_3y_1+x_1y_2-x_2y_1-x_1y_3-x_3y_2}, \\ w=\dfrac{x_2y+xy_1+x_1y_2-x_2y_1-x_1y-xy_2}{x_2y_3+x_3y_1+x_1y_2-x_2y_1-x_1y_3-x_3y_2}. \end{cases} \tag{2}$$

For the point to be inside the triangle, the values of u, v, and w must be between 0 and 1; otherwise, one or two of them will be either larger than 1 or smaller than 0.

According to Eq. (2), it can be determined that such a ray tracing technique is applicable only for a 2D space. However, for most complex geometrics, both rays and triangles should be in a 3D space, and thus we must map them into the 2D space as shown in Fig. 1. Here, a set of parallel rays is used and a 3D vector is mapped into a 2D space by removing one of its components x, y, and z from the triangle vertexes. Note that some surfaces may be degenerated into lines under such circumstance. Therefore, a degenerated detection step should be taken after the mapping process. The detection function is derived as follows:

At first, the triangle is converted into two independent vectors, as described by

$$\begin{cases} \boldsymbol{v}_1 = \boldsymbol{A} - \boldsymbol{B}, \\ \boldsymbol{v}_2 = \boldsymbol{A} - \boldsymbol{C}. \end{cases} \tag{3}$$

Then, we determine the corner of the two vectors by

$$C(\boldsymbol{v}_1, \boldsymbol{v}_2) = \arccos \frac{\|\boldsymbol{v}_1 \cdot \boldsymbol{v}_2\|}{\|\boldsymbol{v}_1\| \cdot \|\boldsymbol{v}_2\|}, \tag{4}$$

where $\|\boldsymbol{v}\|$ is the length of vector $\boldsymbol{v}$, defined as

$$\|\boldsymbol{v}(x, y)\| = \sqrt{x^2 + y^2}, \tag{5}$$

and

$$\boldsymbol{v}_1(x_1, y_1)\cdot\boldsymbol{v}_2(x_2, y_2)=x_1x_2+y_1y_2. \tag{6}$$

Note that in some extreme cases, a triangle may be parallel with the tracing ray, and then vector $\boldsymbol{v}_1$ or $\boldsymbol{v}_2$ may be zero. Under such circumstances, the corner should be defined as zero.

By taking the numerical accuracy of FDTD meshing into account, we introduce an error tolerance ε; i.e., the triangle is invalid when $|C|\leq\varepsilon$ (10^{-7} in current version). Therefore, this triangle should be dropped, and this is acceptable because it usually occurs when the triangle is nearly parallel with the tracing ray. As explained later in Section 2.3, we can properly capture this triangle in another direction, since one triangle surface cannot be parallel with x-, y-, and z-direction at the same time.

2.2 Parallel FDTD meshing method

The complete parallel FDTD meshing process is presented as follows:

Collect intersection points: Obtain intersection points of the ray with each triangle piece, as it is for an individual element and suitable for executing this procedure in parallel.

Snip with FDTD grid: Use one algorithm to snip all intersection points with the FDTD mesh. As shown in Fig. 2a, the traditional method for snipping is just to select the grid point which is the nearest to the intersection point. There are two of three components at (x, y, z) fixed by the ray, but only one is calculated.

Collect interception points and generate mesh: For an empty shell structure in Fig. 2b, we just collect all intersection points. For a solid object, however, a ray may penetrate it several times; i.e., it must Enter->Leave->Enter->…->Enter->Leave the object. Each entering time corresponds to a leaving time, and we split all intersection points into a tuple of (Enter, Leave) pairs. As shown in Fig. 2c, the ray enters the object from the second grid, and leaves it from the fifth grid, and we record it as (2, 5). Then, we expand each tuple into a grid sequence.

The calculations for all interception points here are independent of one another, and this procedure can be naturally parallelized. As we obtain all interception information for the entire model with multiple parallel rays, the structure mesh can be completely reconstructed.

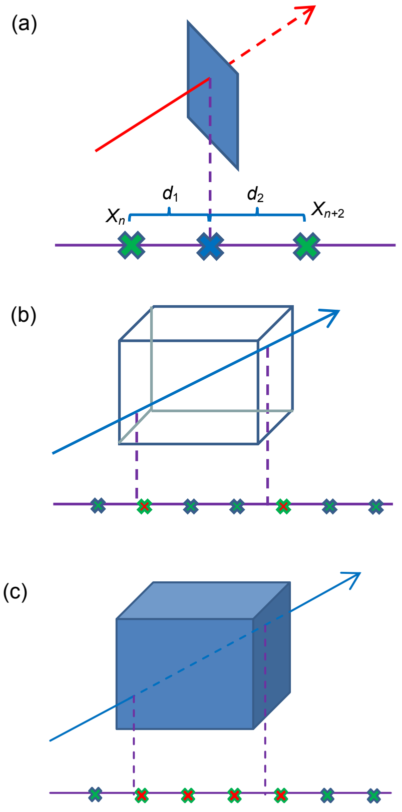

Fig. 2 Schematic of the ray tracing technique
(a) Single ray tracing for a surface patch; (b) Multiple ray tracings for an empty shell structure; (c) Multiple ray tracings for a solid structure

2.3 Meshing for special cases

2.3.1 Parallel surface problem

When the model has some surfaces that are parallel with the rays, the ray tracing may provide wrong mesh information for the structure, as it may result in no or a large number of intersection points (Fig. 3a). So, we need to change the ray direction as shown in Fig. 3b. For a complex 3D model, we always penetrate it using a set of parallel rays with multiple directions so as to improve the meshing accuracy. As mentioned in Section 2.1, we use the orthogonal *x*-, *y*-, and *z*-direction as the tracing approach to simplify the 3D-2D mapping process.

2.3.2 Edge problem

The edge problem may occur when using the searching-based parity count method (Hill, 1996); i.e., as the ray grazes the model edge, the wrong mesh information is generated (Fig. 4a). However, our proposed method does not have such a problem (Fig. 4b). When there is only one interception point, it will be simply excluded.

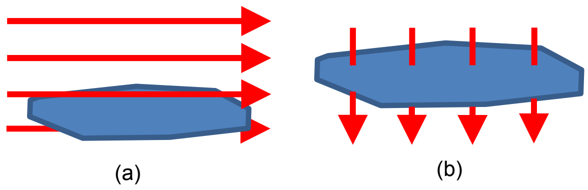

Fig. 3 Rays are parallel with (a) or perpendicular to (b) the surface to be meshed

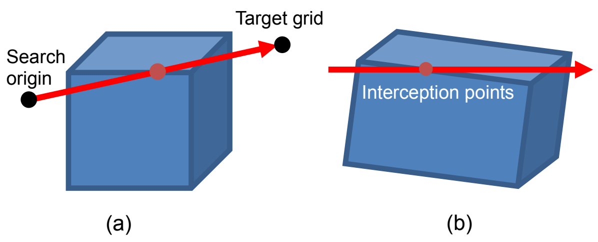

Fig. 4 Wrong (a) and correct (b) FDTD meshing for a 3D structure

2.4 Conformal meshing technique

We know that the traditional searching-based meshing algorithm often uses model facets to build up its mesh. In conformal FDTD simulation, as will be introduced in Section 4, the location of the interception point should also be characterized, while the conventional meshing process just ignores them. Actually, its realization is very complex, since one Yee's cell has 24 independent edges, and the ray tracing technique must be applied individually for each one to calculate the distance between the interception point and the end point of the edge. A 2D example is shown in Fig. 5, where the interception points and distances for all edges are recorded.

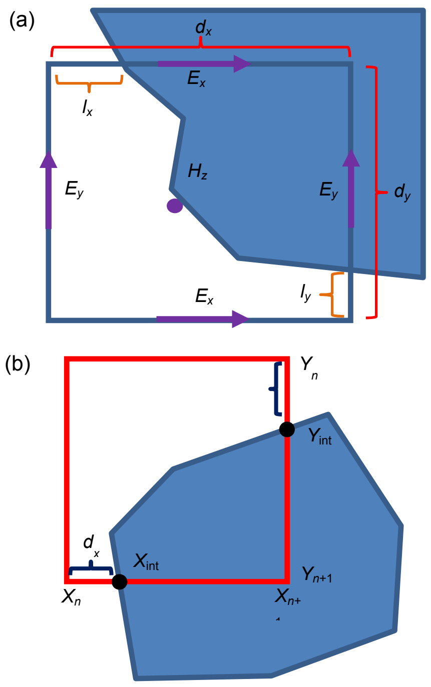

Fig. 5 Schematic of the conformal meshing technique
(a) A Yee-cell in the conformal FDTD algorithm; (b) The distance between the interception point and the endpoint being recorded during conformal meshing

3 Parallelized implementation

Most FDTD algorithms are often realized using the traditional languages of C and Fortran. For a parallel FDTD simulation, however, the function language should provide a better choice, where all variables are unchangeable, i.e., immutable. Here, we use the Microsoft F# (Microsoft, USA) to show its difference from the traditional C language.

3.1 Domain decomposition method using function language

To achieve FDTD parallelization, its iteration domain has to be at first decomposed. For the C language, the general FDTD iteration code is written as

```
for (i=0; i<IMax; i++)
    for (j=0; j<JMax; j++)
      for (k=0; k<KMax; k++)
      // Some iteration code here
```

However, for the function language F#, the code is written as

```
let grids=seq {for i in 0..IMax do for j in
0..JMax do for k in 0..KMax -> (i, j, k)}
  Seq.iter IterationFunc grids
```

The significant difference here is that the variables *i*, *j*, and *k* are aggregated into a three-item tuple and then transferred together into an iteration function.

For a non-decomposed task, these two versions seem to have little difference. However, when we try to decompose the entire problem domain into several parts, i.e., partition the (*i*, *j*, *k*) sequence, the C language must repeat the code several times as

```
Part 1:
for (i=i1Begin; i<i1End; i++)
  for (j=j1Begin; j<j1End; j++)
    for (k=j1Begin; k<j1End; k++)
      // Some iteration code here
Part 2:
for (i=i2Begin; i<i2End; i++)
  for (j=j2Begin; j<j2End; j++)
    for (k=j2Begin; k<j2End; k++)
      // Some iteration code here
 ...
```

For each part, we must define the range of *i*, *j*, and *k* individually. Thus, it is difficult to decompose the domain using a custom decomposing mode. However, in F# the code can be simplified as

```
let iterFunc grids=Seq.iter IterationFunc grids
do Array.iter gridsArray iterFunc
```

Here, 'iterFunc' is the function for iterating an arbitrary (*i*, *j*, *k*) list, and the variable 'gridsArray' is an array in which each item is a sequence of (*i*, *j*, *k*) tuples. The sequence of (*i*, *j*, *k*) can be produced using different methods. Thus, it is possible to decompose the domain according to the model structure.

3.2 Parallelization using function language

The target for domain decomposition is to parallelize both FDTD meshing and simulation. For the program based on the message passing interface (MPI), the code in C language is written as

```
int pi=MPI_GetRank();
for i, j, k ...
  // Some iteration code here
  MPI_Wait(...); // wait and sync all processes
```

Another normal parallelization technique is OpenMP, and its parallelization code is written as

```
#Pragma parallel for
for (int pi=0; i<PI_MAX; i++)
  for i, j, k ...
    // Some iteration code here
```

These two code snippets indicate that we should at first obtain the parallel identifier, and then write two functions so as to obtain the beginning and ending locations of *i*, *j*, and *k* for each part. We need to manually create a parallel iteration loop to control the parallelization, and the waiting function must be placed in a correct position so as to be synchronous for all processes if MPI is to be used. Functions in F# can be parallelized more efficiently than those in the C language, and a sample code is given as

```
let iterFunc grids=Seq.iter IterationFunc grids
do Array.Parallel.iter gridsArray iterFunc
```

Compared with the code in the previous section, only one word 'Parallel' is added to indicate that all parts should be iterated in parallel. Neither parallel control loop nor waiting function is needed.

3.3 Parallelization for meshing process

In the ray tracing technique, each ray is independent of one another, and each triangle on the model surface is also independent. Thus, the parallelization can be based on the rays or triangles and even considering both. A sample of the final pseudocode is presented as follows:

```
let models=readAllModels inputFile
let meshFunc model=
  let rays=buildRaysForModel model
    let singleRay r=rayTrace r model
    Array.Parallel.collect singleRay rays
let result=Array.Parallel.collect meshFunc
models
```

4 Conformal FDTD using proposed meshing technique

The conformal FDTD method is good at handling arbitrary geometrics (Juntunen and Tsiboukis, 2000; Kong *et al.*, 2012; Wang and Yin, 2013). In the algorithm, the ***E***-field is updated in the conventional way, while the updating equation of the ***H***-field needs to be modified. In a source-free homogeneous isotropic medium, the Faraday law can be expressed as

$$\oint_l \boldsymbol{E}\cdot \mathrm{d}l = -\frac{\mathrm{d}}{\mathrm{d}t}\iint_S B\cdot \mathrm{d}s, \tag{7}$$

where S is the part of area outside of the PEC part in the FDTD cell. The parameter l is the integration tour around the area S. According to Wang and Yin (2013), we can obtain the updating equation for the ***H***-field:

$$H_x\Big|_{i,j+1/2,k+1/2}^{n+1/2} = H_x\Big|_{i,j+1/2,k+1/2}^{n-1/2} + \frac{\Delta t}{\mu\cdot S_x(i,j,k)} \cdot\begin{pmatrix} E_y\Big|_{i,j+1/2,k+1}^{n}\cdot l_y\Big|_{i,j,k+1}^{n} - E_y\Big|_{i,j+1/2,k}^{n}\cdot l_y\Big|_{i,j,k}^{n} \\ -E_z\Big|_{i,j+1,k+1/2}^{n}\cdot l_z\Big|_{i,j+1,k}^{n} - E_z\Big|_{i,j,k+1/2}^{n}\cdot l_z\Big|_{i,j,k}^{n}\end{pmatrix}. \tag{8}$$

Here, $l_x\Big|_{i,j,k}^{n}$ and $S_x(i, j, k)$ should be recorded during the meshing process. Equation in the y- and z-direction can be similarly obtained. In an improved conformal FDTD (Kong *et al.*, 2012; Wang and Yin, 2013) method, the permittivity of free space is changed into diag(ε_x, ε_y, ε_z), and a Courant coefficient q is defined. The CFL stability condition is revised as

$$\Delta t \le q / c\cdot\sqrt{1/\left(\varepsilon_x\Delta x^2\right)+1/\left(\varepsilon_y\Delta y^2\right)+1/\left(\varepsilon_z\Delta z^2\right)}. \tag{9}$$

A pair of parameters z_y=$\Delta x/\Delta y$ and z_z=$\Delta y/\Delta z$ are introduced, and a spatial resolution coefficient is given by

$$R=\lambda / \sqrt{\Delta x^2+\Delta y^2+\Delta z^2}, \tag{10}$$

where λ=c/f+f_0, and f_0 is the frequency at which the numerical dispersion error reaches its minimum. A set of parameters q, R, z_y, and z_z were also proposed in Wang and Yin (2013). Let

$$\begin{cases} A_{\max} = \pi\Big/\left[3R\arcsin\left(\dfrac{1}{\sqrt{3}}\sin\dfrac{\pi}{\sqrt{3}R}\right)\right], \\ K_1 = \pi\Big/\left[R\sqrt{1+1/Z_y^2+1/Z_z^2}\right], \\ a=\sin^2\dfrac{K_1}{1-0.5(A_{\max}-1)} \\ \quad \cdot\left\{Z_y^2\sin^2\dfrac{K_1}{Z_y[1-0.5(A_{\max}-1)]}\right\}^{-1}, \\ b=\sin^2\dfrac{K_1}{1-0.5(A_{\max}-1)} \\ \quad \cdot\left\{Z_z^2\sin^2\dfrac{K_1}{Z_y[1-0.5(A_{\max}-1)]}\right\}^{-1}, \\ K_2=\sqrt{1+aZ_y^2+bZ_z^2}. \end{cases} \tag{11}$$

Therefore, the best solution for ε_x, ε_y, and ε_z are (Kong *et al.*, 2012)

$$\begin{cases} \varepsilon_x = K_2/(K_1 q\sqrt{ab}) \\ \quad \cdot\arcsin\{q/K_1\sin[K_1/(1-0.5Q)]\}, \\ \varepsilon_y = a\cdot\varepsilon_x, \\ \varepsilon_z = a\cdot\varepsilon_x. \end{cases} \tag{12}$$

The E_x-field should be updated by

$$\begin{aligned} E_x^{n+1}(i+1/2,j,k) &= E_x^{n}(i+1/2,j,k) \\ &+\frac{\Delta t}{\varepsilon_x\Delta y}\cdot\begin{bmatrix} H_z^{n+1/2}(i+1/2,j+1/2,k) \\ -H_z^{n+1/2}(i+1/2,j-1/2,k)\end{bmatrix} \\ &-\frac{\Delta t}{\varepsilon_x\Delta z}\cdot\begin{bmatrix} H_z^{n+1/2}(i+1/2,j+1/2,k) \\ -H_z^{n+1/2}(i+1/2,j-1/2,k)\end{bmatrix}, \end{aligned} \tag{13}$$

and both E_y- and E_z-field can be obtained following a similar process (suppressed here).

4.1 Surface current calculation

Based on the boundary condition of current continuity on the PEC surface, the current at point ***P*** can be calculated by

$$\boldsymbol{J}(\boldsymbol{P}) = \boldsymbol{n}(\boldsymbol{P})\times\boldsymbol{H}(\boldsymbol{P}) = \begin{pmatrix} e_x & e_y & e_z \\ n_x & n_y & n_z \\ H_x(\boldsymbol{P}) & H_y(\boldsymbol{P}) & H_z(\boldsymbol{P})\end{pmatrix}, \tag{14}$$

where $\boldsymbol{e}$ is a unit vector, $\boldsymbol{n}$ is the normal vector at $\boldsymbol{P}$, and $\boldsymbol{H}$ is the corresponding magnetic field. It can be expressed as a weighted sum of the neighbor surface normal vectors, and for an arbitrary surface, we obtain

$$\boldsymbol{n}_c = \begin{pmatrix} x_3 - x_2 \\ y_3 - y_2 \\ z_3 - z_2 \end{pmatrix} \times \begin{pmatrix} x_1 - x_2 \\ y_1 - y_2 \\ z_1 - z_2 \end{pmatrix}. \tag{15}$$

Thus, the normal vector $\boldsymbol{n}_v$ of a triangle vertex v is calculated by

$$\boldsymbol{n}_v = \sum_1^N \alpha^i \cdot \boldsymbol{n}_c^i, \tag{16}$$

where $\boldsymbol{n}_c^i$ is the surface vector for triangle i, and α^i is the angle between triangle i and the target vertex.

As FDTD uses the discrete Yee cell, H_x-, H_y-, and H_z-components are separated into different locations (Fig. 6). We need to use an interception technique to obtain their values at the center location. For the H_x-component, we obtain

$$\begin{cases} \boldsymbol{Q}_{11} = (i+1, j+1/2, k+1/2), \\ \boldsymbol{Q}_{22} = (i+1, j+3/2, k+3/2), \\ \boldsymbol{Q}_{12} = (i+1, j+1/2, k+3/2), \\ \boldsymbol{Q}_{21} = (i+1, j+3/2, k+1/2), \end{cases} \tag{17}$$

$$H_x(\boldsymbol{P}) = \frac{d_{z1}}{\Delta z}\left(\frac{d_{11}}{\Delta y}H_x(\boldsymbol{Q}_{11}) + \frac{d_{12}}{\Delta y}H_x(\boldsymbol{Q}_{12})\right) + \frac{d_{z2}}{\Delta z}\left(\frac{d_{21}}{\Delta y}H_x(\boldsymbol{Q}_{21}) + \frac{d_{22}}{\Delta y}H_x(\boldsymbol{Q}_{22})\right), \tag{18}$$

where $\boldsymbol{Q}_{ij}$ is the location of the neighbour $\boldsymbol{H}$-field surrounding $\boldsymbol{P}$, and d_{ij} is the distance between $\boldsymbol{Q}_{ij}$ and $\boldsymbol{P}$.

In some special cases, one or more neighbour $\boldsymbol{H}$-components for a surface vertex may be located inside the model. Due to shielding effect, the electromagnetic field in the internal space is always zero. Numerically, the value has a mutation on the surface boundary. Under such circumstances, interceptions will be extended into nine grids, and the final one can be rewritten as

$$H_x(\boldsymbol{P}) = \frac{1/d_1}{\sum_{u=1}^N 1/d_u} H_x(\boldsymbol{Q}_1) + \ldots + \frac{1/d_N}{\sum_{u=1}^N 1/d_u} H_x(\boldsymbol{Q}_N), \tag{19}$$

where N is the count of non-zero H_x-component in the nine surrounding grids.

In Kong *et al.* (2012), the weight coefficient d_i was set to be the distance between the point $\boldsymbol{P}$ and the neighbor $\boldsymbol{H}$-field location $\boldsymbol{Q}_i$. However, a more accurate result can be obtained as we set the coefficient d_i as the distance between the centers of the grids where $\boldsymbol{P}$ and $\boldsymbol{Q}_i$ belong to them, respectively.

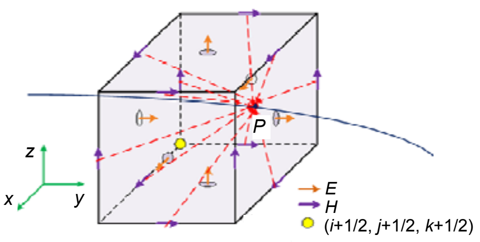

Fig. 6 The point *P* is the one on the model surface at which we want to record its current density

5 Numerical results and discussion

5.1 Mesh algorithm verification

At first, we select a sphere made of eight different material to test our meshing algorithm. Its radius is set to be 1 m. Fig. 7a shows our mesh result, with each material rendered using a unique color. It is evident that the mesh agrees well with the model. We also capture the cross section of this sphere and show the detected conformal edge information in Fig. 7b. The distance between the center of an FDTD cell and the real edge captured by conformal meshing will be recorded during this process. Further, we generate the FDTD mesh of a PEC tank model of 17.4 m×9.3 m×8.5 m to verify our surface mesh result (Fig. 7c), and the mesh also agrees well with the model.

5.2 Parallel efficiency analysis

We further use the tank model (Fig. 7c) to verify the parallel efficiency of our developed parallel FDTD meshing algorithm. Its efficiency can be defined by a 'naturally parallel coefficient' ψ as the

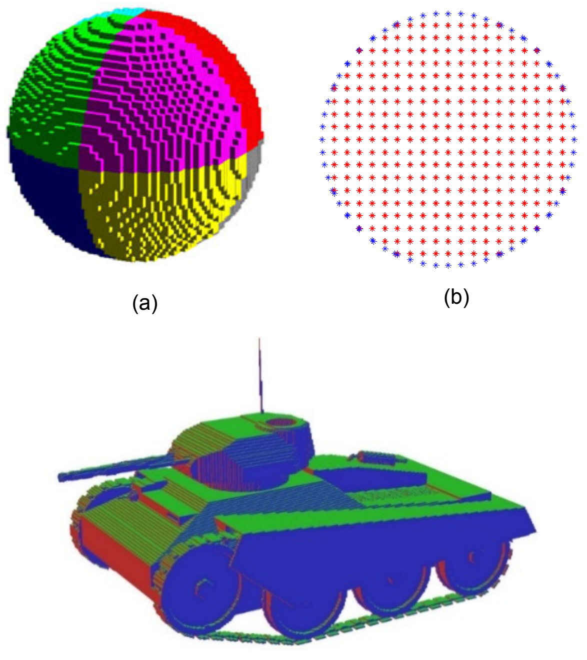

Fig. 7 FDTD meshes
(a) A sphere made of eight different materials; (b) Detected edge information during the conformal meshing process for a cross-section of the sphere; (c) A tank model. The red marks are the centers of the FDTD cells, and the blue marks are the real edges captured by conformal meshing. References to color refer to the online version of this figure

percentage of the execution time that can be paralleled (Hill, 1996).

Table 1 shows the time allocation for each step in the meshing process of a tank model. The value of ψ is the sum of all steps which are parallelizable (Hill, 1996), and it is about 0.85. In most cases, it is difficult to measure. However, the final speed-up S can be easily measured according to $S=T_{NP}/T_P$ (Hill, 1996), where T_{NP} is the time allocation for a non-parallelized version and T_P is the one for a parallelized version. Therefore, its ideal parallel speed-up value can be defined as $S=N_P/[(1-\psi)N_P+\psi]$ (Hill, 1996). Further, the value of ψ can be calculated by $\psi=(N_PS-N_P)/(N_PS-1)$.

Table 2 shows the recorded naturally parallel coefficients for meshing of the tank model. The hardware is a Dell Vostro 260 workstation with 16 GB DDR3-1600 MHz memory and an Intel I5-2400 (4 cores) CPU installed, and the operational system is Windows 8 Enterprise Edition. Our algorithm is implemented using Microsoft F# 3.0 with Visual Studio 2012. In Table 2, S_2 and S_4 are the speed-up values for using two and four CPU cores in the same PC, respectively, while ψ_2 and ψ_4 correspond to their naturally parallel coefficients, respectively.

Table 1 Time cost for each step in the meshing process of a tank model

Step	Parallelizability	Time (s)	Percentage
Read model	No	3.12	7.00%
Build up parallel rays	Yes	0.65	1.46%
Get interception points	Yes	37.24	83.55%
Merge results	No	1.47	3.21%
Output mesh	No	2.09	4.69%

Table 2 Naturally parallel coefficients for meshing the tank model

Mesh size	CPU time (s)			S_2	S_4	ψ_2	ψ_4
	1	2	4				
1.0	13.7	7.7	5	1.779	2.765	0.609	0.702
0.5	38.7	21.5	14.1	1.798	2.738	0.615	0.699
0.1	777.8	424.2	265.3	1.833	2.932	0.625	0.720

It is observed that the naturally parallel coefficient is closely related to the CPU count, and it is about 60% and 70% for two and four CPU cores, respectively, while the coefficient for the traditional searching-based method is only about 47% (Hill, 1996). So, our proposed method has a higher parallel efficiency.

5.3 Conformal FDTD method for surface current calculation

To verify our conformal FDTD algorithm, we first simulate the radar cross section (RCS) of a PEC sphere with 1-m radius, and make a comparison between our proposed algorithm, FEKO software (EMSS, South Africa), and MIE-theory (Fig. 8). We then calculate its surface current distribution and compare it with the FEKO software. The incident wave is a Gaussian pulse and its 3-dB bandwidth is 900 MHz. It is evident that our improved FDTD algorithm achieves good accuracy in capturing the surface current distribution of a 3D PEC object.

We take a J-10 aircraft as another example (Fig. 9a). The model is 6.86 m×3.85 m×1.86 m, and the mesh grid size for all directions is chosen to be 0.01 m in Fig. 9b.We use our developed FDTD algorithm together with the commercial CST (CST, USA) software to simulate its surface current distribution in the presence of a lightning electromagnetic pulse (LEMP), as described by

$$E(t)=E_0k(\mathrm{e}^{-\beta t}-\mathrm{e}^{-\alpha t}), \tag{20}$$

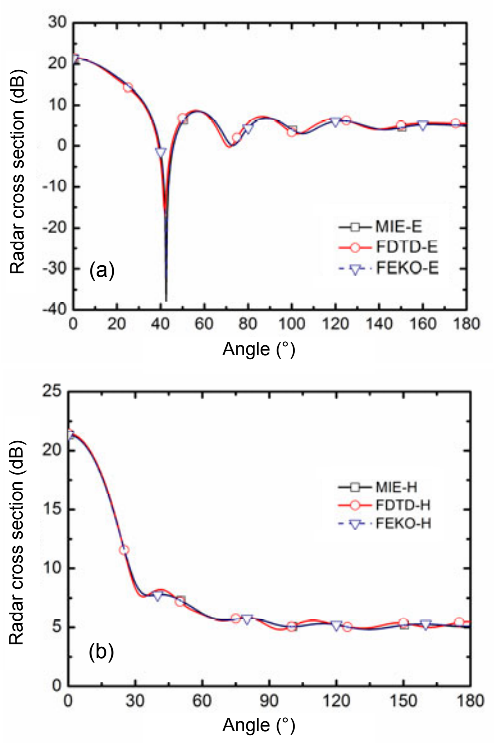

Fig. 8 Simulated RCS and surface current distributions of a PEC sphere with 1-m radius
The RCS comparison of ***E***-plane (a) and ***H***-plane (b) using our proposed algorithm, FEKO, and MIE-theory, respectively

where E_0 is the peak field value, k is a fixed coefficient, α and β are parameters to describe the rising- and falling-edge of the LEMP. Here we choose k=1.06, α=4.76×10^6 s^{-1}, and β=4.0×10^6 s^{-1}. The incident electromagnetic pulse is from the +Z direction and is polarized in the +Y direction. Fig. 9c shows that the predicted surface current distribution of the aircraft obtained using our algorithm agrees very well with that of the CST (Fig. 9d).

The third example is the PEC tank model as shown in Fig. 7c. The FDTD grid size is chosen to be 1/40 m, and the incident wave is the same as that used in Fig. 10. Figs. 11a and 11b show the steady surface current distribution when the incident LEMP comes from +Z and +Y directions, respectively.

Table 3 shows the comparison for the recorded maximum surface current density between the results obtained using the developed FDTD and those of the commercial CST software. The position of the maximum surface current is marked in Figs. 10 and 11 with a red circle. We would like to say that our algorithm can achieve good accuracy compared with CST software.

Finally, we use our parallelized algorithm to calculate the surface current distribution on an aircraft carrier with 20 aircrafts. A surface current view is shown in Fig. 12. It is observed that our algorithm

Table 3 The maximum current density comparison for different models

Model	Maximum current density using CST (A/m^2)	Maximum current density using FDTD (A/m^2)	Relative error
J-10	0.0372	0.0397	6.72%
Tank	45.2	44.1	2.43%

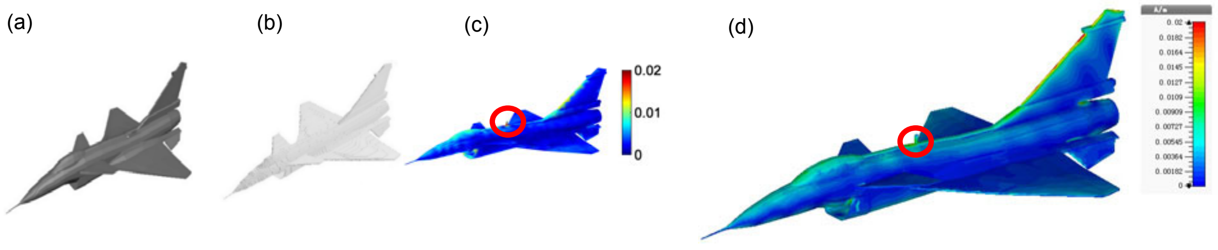

Fig. 9 Simulated surface current distribution of a J-10 aircraft at 300MHz in the presence of an LEMP incidence from the +*Z* direction, and polarized in the +*Y* direction
(a) Original 3DS-Max model of the J-10 aircraft; (b) Its mesh grid view; (c) Surface current distributions obtained by our algorithm; (d) Surface current distributions obtained by the commercial software CST. The maximum surface current is recorded at the central breakout, as marked with a circle. References to color refer to the online version of this figure

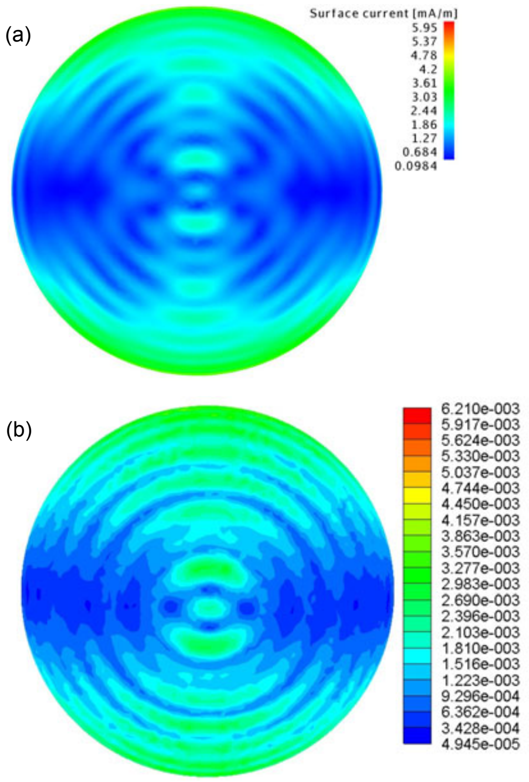

Fig. 10 The surface current obtained by the commercial FEKO software (a) and our improved algorithm (b)

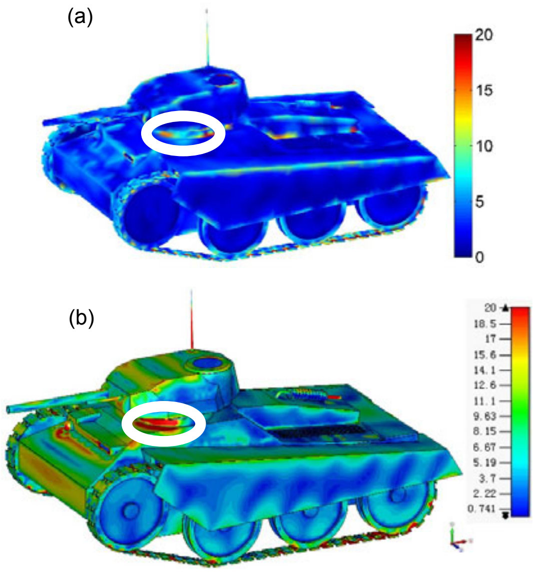

Fig. 11 Simulated surface current distribution of a PEC tank model at 200 MHz in the presence of an LEMP incidence from the +*Z* direction, and polarized in the +*Y* direction

(a) Our algorithm; (b) Commercial CST software. The maximum surface current occurred at the connection point between the tank body and header, as marked with a circle. References to color refer to the online version of this figure

works well with high efficiency. The scale and simulation information of such an extremely large structure is shown in Table 4.

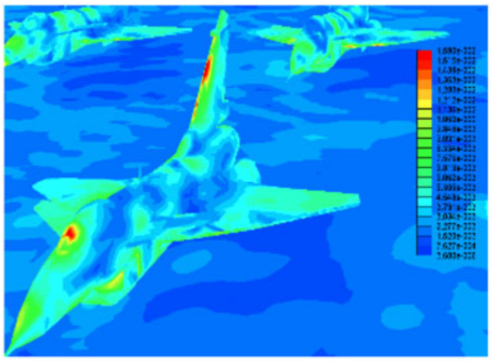

Fig. 12 Simulated surface current distribution at 300 MHz for a large mother board with 20 aircrafts carried (here only one aircraft is shown)

Table 4 Problem scale and simulation information for the mother board model with 20 aircrafts

Term	Specification
Domain scale	328 m×72 m×59 m
Mesh size	0.1 m×0.1 m×0.1 m
Target frequency	300 MHz
Memory	About 80 GB
CPU	48 cores/single core
Run time	About 30 h/more than one week (unfinished)

6 Conclusions

In this paper, a complete parallel meshing method based on the ray racing technique has been proposed with different meshes demonstrated, such as regular and conformal ones. We have employed a new computer language, Microsoft F#, for algorithm coding to greatly take its parallelization advantage. An improved conformal FDTD algorithm, integrated with an improved surface current algorithm, has been presented with higher simulation accuracy obtained. In particular, we have tested its efficiency and capability by simulating a sphere ball made of eight different materials, a PEC tank model, an electrically large J-10 aircraft model, and an aircraft carrier with 20 aircrafts, respectively. Both surface current and field distributions have been successfully captured in the presence of an electromagnetic pulse.

References

Benkler, S., Chavannes, N., Kuster, N., 2008. Mastering conformal meshing for complex CAD-based C-FDTD simulations. *IEEE Antennas Propag. Mag.*, **50**(2):45-57. [doi:10.1109/MAP.2008.4562256]

Flubacher, R., Luebbers, R., 2003. FDTD mesh generation using computer graphics technology. IEEE Antennas and Propagation Society Int. Symp., p.333-336. [doi:10.1109/APS.2003.1217464]

Guiffaut, C., Mahdjoubi, K., 2001. A parallel FDTD algorithm using the MPI library. *IEEE Antennas Propag. Mag.*, **43**(2):94-103. [doi:10.1109/74.924608]

Hadi, M.F., Mahmoud, S.F., 2007. Optimizing the compact-FDTD algorithm for electrically large waveguiding structures. *Prog. Electromagn. Res.*, **75**:253-269. [doi:10.2528/PIER07060703]

Hill, J., 1996. Efficient Implementation of Mesh Generation and FDTD Simulation of Electromagnetic Fields. MS Thesis, Worcester Polytechnic Institute, MA, USA.

Hsu, H.T., Kuo, F.Y., Chou, H.T., 2009. Convergence study of current sampling profiles for antenna design in the presence of electrically large and complex platforms using FIT-UTD hybridization approach. *Prog. Electromagn. Res.*, **99**:195-209. [doi:10.2528/PIER09092404]

Juntunen, J.S., Tsiboukis, T.D., 2000. Reduction of numerical dispersion in FDTD method through artificial anisotropy. *IEEE Trans. Microw. Theory Tech.*, **48**(4):582-588. [doi:10.1109/22.842030]

Kim, J., Teixeira, F.L., 2011. Parallel and explicit finite-element time-domain method for Maxwell's equations. *IEEE Trans. Antennas Propag.*, **59**(6):2350-2356. [doi:10.1109/TAP.2011.2143682]

Kong, L.Y., Wang, J., Yin, W.Y., 2012. A novel dielectric conformal FDTD method for computing SAR distribution of the human body in a metallic cabin illuminated by an intentional electromagnetic pulse (IEMP). *Prog. Electromagn. Res.*, **126**:355-373. [doi:10.2528/PIER11112702]

Lei, J.Z., Liang, C.H., Ding, W., *et al.*, 2008. EMC analysis of antennas mounted on electrically large platforms with parallel FDTD method. *Prog. Electromagn. Res.*, **84**:205-220. [doi:10.2528/PIER08071303]

Shan, X., Guan, S., Liu, Z., *et al.*, 2013. A new energy harvester using a piezoelectric and suspension electromagnetic mechanism. *J. Zhejiang Univ.-Sci. A (Appl. Phys. & Eng.)*, **14**(12):890-897. [doi:10.1631/jzus.A1300210]

Srisukh, Y., Nehrbass, J., Teixeira, F.L., *et al.*, 2002. An approach for automatic grid generation in three-dimensional FDTD simulations of complex geometries. *IEEE Antennas Propag. Mag.*, **44**(4):75-80. [doi:10.1109/MAP.2002.1043151]

Taflove, A., Hagness, S.C., 2000. Computational Electrodynamics: the Finite-Difference Time-Domain Method (2nd Ed.). Artech House, Norwood, MA, USA.

Vaccari, A., Lesina, A.C., Cristoforetti, L., *et al.*, 2011. Parallel implementation of a 3D subgridding FDTD algorithm for large simulations. *Prog. Electromagn. Res.*, **120**:263-292.

Wang, H., Tang, L., Guo, Y., *et al.*, 2014. A 2DOF hybrid energy harvester based on combined piezoelectric and electromagnetic conversion mechanisms. *J. Zhejiang Univ.-Sci. A (Appl. Phys. & Eng.)*, **15**(9):711-722. [doi:10.1631/jzus.A1400124]

Wang, J., Yin, W.Y., 2013. Development of a novel FDTD (2, 4)-compatible conformal scheme for electromagnetic computations of complex curved PEC objects. *IEEE Trans. Antennas Propag.*, **61**(1):299-309. [doi:10.1109/TAP.2012.2216851]

Xiong, R., Chen, B., Han, J.J., *et al.*, 2012. Transient resistance analysis of large grounding systems using the FDTD method. *Prog. Electromagn. Res.*, **132**:159-175. [doi:10.2528/PIER12082601]

Yang, M., Chen, Y., 1999. AutoMesh: an automatically adjustable, nonuniform, orthogonal FDTD mesh generator. *IEEE Antennas Propag. Mag.*, **41**(2):13-19. [doi:10.1109/74.769687]

Yu, W.H., Mittra, R., 2000. A conformal FDTD software package modeling antennas and microstrip circuit components. *IEEE Antennas Propag. Mag.*, **42**(5):28-39. [doi:10.1109/74.883505]