John L. Volakis
Rad. Lab., EECS Dept.
University of Michigan
Ann Arbor, MI 48109-2122
(734) 647-1797
(734) 647-2106 (Fax)
volakis@umich.edu (email)

David B. Davidson
Dept. E&E Engineering
University of Stellencosch
Stellenbosch 7600, South Africa
(+27) 21 808 4458
(+27) 21 808 4981 (Fax)
davidson@ing.sun.ac.za (e-mail)

Foreword by the Editors

As anyone who has tried to enter a complex, real-world geometry (as opposed to canonical shapes, such as spheres, cylinders, etc.) into a CEM program can attest, this can be a very frustrating, time-consuming, and error-prone operation. Quite often, data is already available in some other CAD format, and if this can be used, the process can be automated, either fully or at least to some extent. This month's contribution discusses this type of operation for an FDTD code. The authors discuss some computer-graphics algorithms used in the process of converting an AutoCAD© description into FDTD "cells." The examples given – an M1 main battle tank and a Scud transporter/erector/launcher weapons system – certainly qualify as examples of complex geometries! Computer graphics is a field that CEM programmers quite often need, but of which they are often fairly ignorant, and it is not usually discussed in texts on the subject. We trust that this contribution will be useful for CEM-code and model developers.

An Approach for Automatic Grid Generation in Three-Dimensional FDTD Simulations of Complex Geometries

Y. Srisukh, J. Nehrbass, F. L. Teixeira, J.-F. Lee, and R. Lee

ElectroScience Laboratory and Department of Electrical Engineering
The Ohio State University, Columbus OH 43212
Tel: +1 (614) 247-6499; Fax: +1 (614) 292-7297; E-mail: teixeira@ee.eng.ohio-state.edu

Abstract

The Finite-Different Time-Domain (FDTD) method has become a popular method in computational electromagnetics because of its simplicity, flexibility, and efficiency. However, the process of generating a three-dimensional (3D) FDTD grid can be time-consuming and error-prone when manually manipulating complex geometries. In order to expedite the generation of FDTD grids, computer-graphics-based methods can alternatively be used. Starting from the geometric description of the problem domain given by a CAD file, an FDTD grid with a specific spatial resolution can be automatically produced. A simple algorithm to this end is discussed in this paper, along with sample results.

Keywords: FDTD methods; computer graphics; computer input-output; design automation; computational geometry

1. Introduction

The Finite-Difference Time-Domain (FDTD) method is a widely used numerical method for solving electromagnetic propagation and scattering problems [1]. However, the application of the FDTD method to problems involving complex geometries often requires a costly and error-prone process of manually creating the FDTD grid. In this paper, we will discuss an automatic grid-generation technique to produce three-dimensional (3D) FDTD grids of complex geometries directly from input CAD files. This automatic technique both reduces the cost associated with manual grid generation, and minimizes the possibility of errors in the geometric depiction of the objects.

The focus of the automatic FDTD grid-generation algorithms described in this paper is for three-dimensional applications. The main object for this paper can be summarized in Figure 1, where a two-dimensional view of simple objects and the associated FDTD grids are shown. In the algorithm implementation, we assume that the objects inside the problem domain are bounded. We also assume uniform cells in the three-dimensional FDTD grid, i.e., $\Delta x = \Delta y = \Delta z$, for simplicity.

2. CAD Description of Objects in the Form of Facets

In most CAD software, objects are simply described in terms of nodes, and facets formed by the connection between nodes. Therefore, in order to efficiently use CAD files as input, the automatic FDTD grid-generation algorithm described here assumes this type of description for all volumetric objects inside the FDTD domain. In particular, the sample results shown in Section 4 of this paper use input CAD files generated by the AutoCAD© software. However, the algorithm itself is not limited to use with just the AutoCAD software. Other CAD software can also be used.

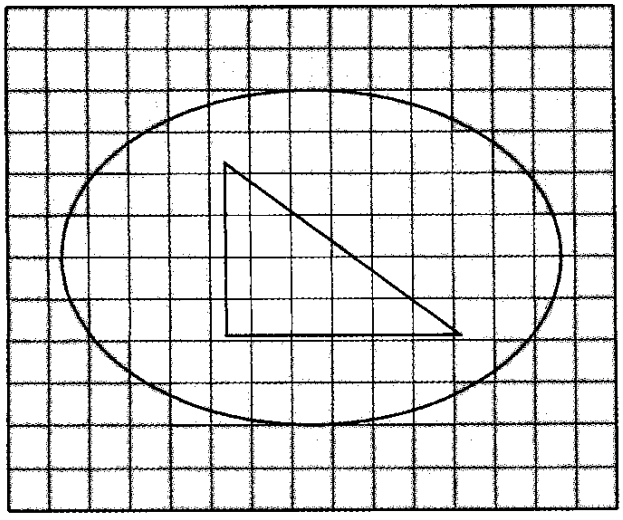

Figure 1. A two-dimensional depiction of automatic FDTD grid generation.

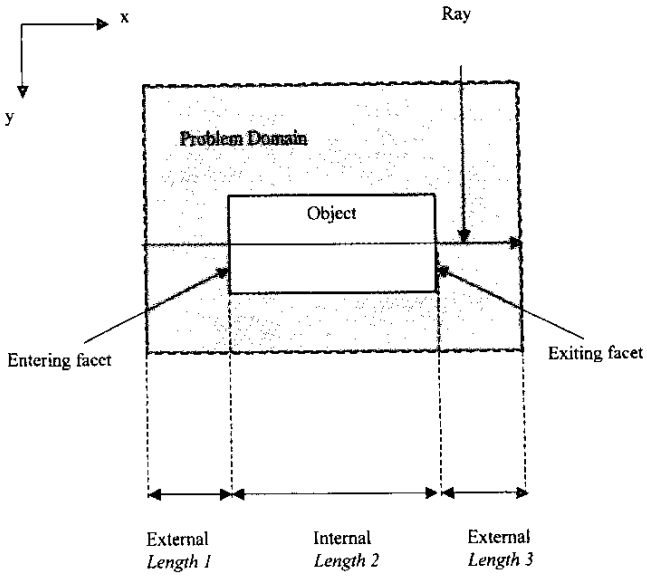

Figure 2a. A problem domain with one object.

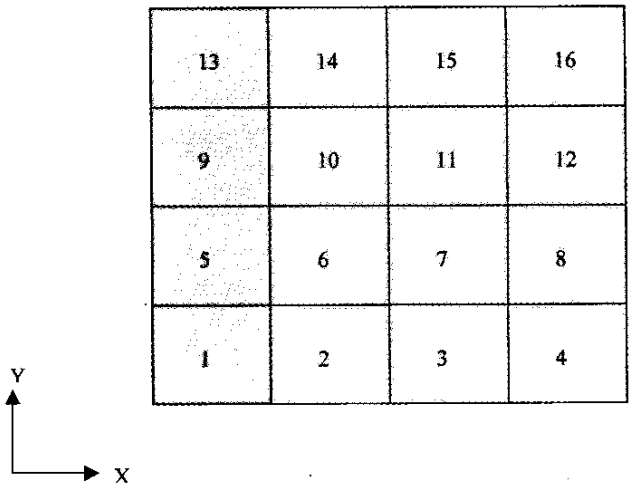

Figure 2b. A layer of cells and their indexes in the *xy* plane of a 4 × 4 × 4 FDTD grid with the tracing rays in the *y* direction.

3. The Algorithm

We assume a rectangular three-dimensional FDTD domain, and define a global Cartesian coordinate system, aligned with the FDTD-domain boundaries. In order to speed up the grid-generation process, the FDTD grid of a problem domain is set up by defining *cell columns* in a given scanning direction. This means that instead of generating individual cells, a tracing direction is first selected along one of the main axes, *x*, *y*, or *z*, and then cell columns are produced in that direction. Usually, this direction corresponds to the largest dimension of the problem domain, in order to minimize the computational cost. Each cell column represents either an interior object (interior cells), or the exterior to all the objects inside the problem domain (exterior cells, usually corresponding to free space). Therefore, an important part of creating the three-dimensional FDTD grid is the identification and classification of the interior cells, i.e., the appropriate cell columns that represent each object inside the problem domain. Methods borrowed from com-

puter graphics can be used for this purpose. To identify the cell columns, two algorithms, ray crossing and winding-number computation, are employed in sequence.

3.1 Grouping of Cells

To generate the FDTD grid in an automatic manner, the problem domain is first scanned by shooting a series of tracing rays from the lower limit of the problem domain to the upper limit of the problem domain. The number of rays equals the number of cells on the cross section perpendicular to the shooting direction. A graphical representation of this process is in shown in Figure 2.

Figure 2a shows a two-dimensional cross section of a problem domain, with a simple square object inside the problem domain. In this case, the tracing direction is chosen as the x direction and, hence, the columns of cells are also produced in that direction. The tracing ray in Figure 2a passes through the object by first entering the facet denoted as the *entering facet*, and then exiting the object through the facet denoted as the *exiting facet*. By identifying the entering coordinates and the exiting coordinates, three lengths can be calculated from the tracing process: in this case,

$$\begin{aligned} Length\_1 &= X_1 - X_0, \\ Length\_2 &= X_2 - X_1, \\ Length\_3 &= X_3 - X_2, \end{aligned} \tag{1}$$

where X_0 is the x coordinate of the lower limit of the problem domain, X_1 is the x coordinate of the entering coordinates on the entering facet, X_2 is the x coordinate of the exiting coordinates on the exiting facet, and X_3 is the x coordinate of the upper limit of the problem domain. If the tracing direction is in the y direction or the z direction, the coordinates used in Equation (1) would change to the y coordinate or the z coordinate, accordingly.

In this particular example, the cells can be classified into three cell columns, derived by these three lengths, using the following equation:

$$cells(Length) = \text{int}\left(\frac{Length}{gridsize}\right). \tag{2}$$

From Equation (2), the absolute lengths are divided by the (uniform) cell size, and then rounded to the closest integer, which represents the number of cells representing the length. Notice that the indexing of the cells is important. The starting cell and the ending cell of adjacent lengths must be correctly indexed; otherwise, the indices of all the cells that come afterwards can be shifted. For instance, in the example mentioned earlier, if $Length\_1$ is represented by six cells and the index of the starting cell of $Length\_1$ is 5, then the index of the ending cell must be 10, and the starting cell of the next length, $Length\_2$, must have index 11. In the cases when the tracing rays are not in the x direction, the indices of cells in cell columns will not be sequential. For example, in Figure 2b, consider a $4 \times 4 \times 4$ FDTD grid, with the tracing rays in the y direction (cell columns in the y direction). If the starting cell's index of a cell column is 1, the index of the adjacent cell to the starting cell in that cell column will be 5. Another important check while generating the cell columns from the lengths is the number of cells allowed in each cell column. In other words, the total number of cells in all columns cells must be equal to the total number of cells allowed in the x direction.

For problems with more than one object inside the domain, the tracing-ray scheme works in a similar way, except that each different object is distinguished by using different object-number classifiers for the corresponding cells. In this manner, objects with different material parameters can also be easily distinguished

3.2 Ray-Crossing Algorithm

In the process of grouping the cells, the intersection points of the scanning ray and the facets on the objects must be identified. The ray-crossing algorithm is used to identify the crossing points. First, we need to find out whether or not the tracing ray crosses the extended plane to which a given facet belongs (the *facet plane*). This can be determined by calculating the ratio

$$\chi = \frac{\mathbf{n} \cdot \mathbf{t}}{\mathbf{n} \cdot \mathbf{l}}, \tag{3}$$

where $\mathbf{n}$ is a unit vector in the normal direction of the facet, $\mathbf{t}$ is a vector from the first end of the tracing ray to the first point of the testing facet, and $\mathbf{l}$ is a vector from the first end of the tracing ray to the second end of the tracing ray. The tracing ray intersects the facet plane if χ lies between 0 and 1 ($0 < \chi < 1$). A graphical depiction of the vectors in Equation (3) is shown in Figure 3 (note that $\mathbf{n}$ and $\mathbf{l}$ are not necessarily parallel).

If the tracing ray crosses the facet plane, the coordinate of the crossing point on this plane, along the ray-tracing direction, can be calculated from

$$\begin{aligned} Crosspoint_x &= l_{1x} + \chi l, \\ Crosspoint_y &= l_{1y} + \chi l, \\ Crosspoint_z &= l_{1z} + \chi l, \end{aligned} \tag{4}$$

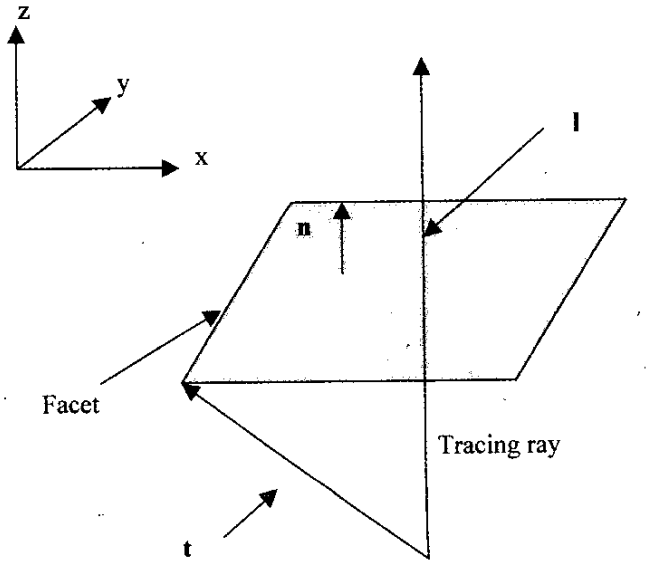

Figure 3. A graphical sample of the vectors in Equation (3).

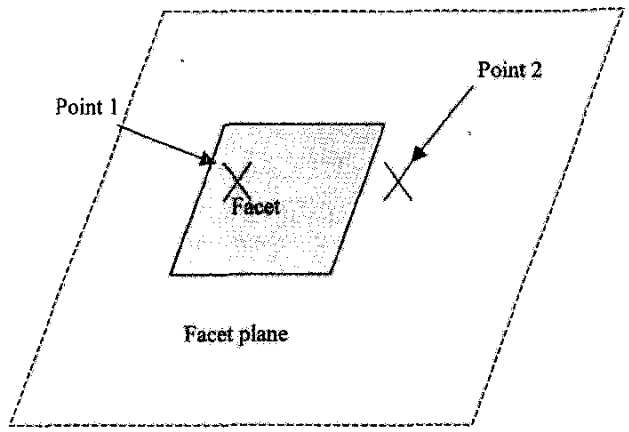

Figure 4. Two possibilities for crossing points on the testing-facet plane.

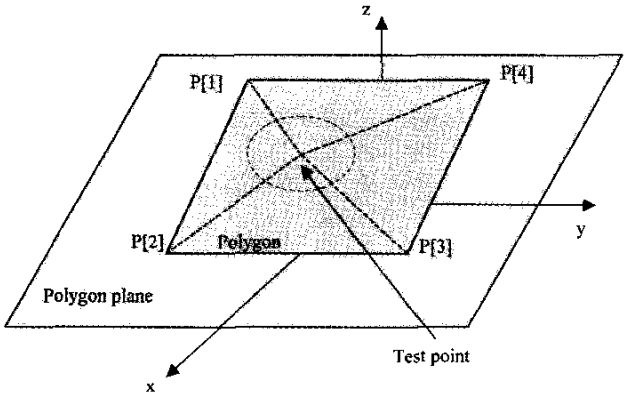

Figure 5. An example of the winding-number calculation.

where $Crosspoint_{x,y,z}$ are the x, y, and z coordinates of the crossing point on the facet plane; $l_{1x,y,z}$ are the x, y, and z coordinates of the first point of the **l** vector, and l is the length of the **l** vector.

Next, the algorithm determines whether or not the ray-crossing point actually lies inside the testing facet. Figure 4 shows two possibilities for crossing points on the testing-facet plane. In this figure, both Point 1 and Point 2 lie on the testing-facet plane, but only Point 1 lies inside the testing facet. To distinguish between the two cases, a winding-number calculation is used, as described next.

3.3 Winding-Number Computation

The winding-number computation can be applied in either two or three dimensions. For three-dimensional problems, it can be applied to check whether or not a point lies inside a polyhedron. For a two-dimensional case, the winding-number calculation can be used to check if a point lies in a polygon. The two-dimensional winding-number calculation will be applied here to check whether or not a testing point lies inside a testing facet.

Consider the example depicted in Figure 5, showing a parallelogram and a testing point. The testing point is on the plane of the polygon. By drawing line segments from the testing point to the vertices of the polygons, four angles are produced by successive pairs of segments. For example, in Figure 5, the line from the testing point to vertex P[1] and the line from the testing point to vertex P[2] form such an angle. Let us denote these angles as *vertex-pair angles*, or α_i. The total number of vertex-pair angles, n, is equal to the number of vertices of the polygon. The winding number, W, is then simply given by the sum

$$W = \sum_{i=1}^{n} \alpha_i , \quad i = 1,2,...,n . \tag{5}$$

The testing point lies inside the polygon if and only if $W = 2\pi$.

This two-dimensional winding-number calculation works well, except for the case where the testing points lie inside the testing facets but very close to or right at the edges or vertices of the testing facets. An additional checking is performed for those cases, as described next.

3.4 Additional Checking

3.4.1 Edge Points

For cases where a testing point lies very close to or right at an edge of a testing facet, we first pick a vertex of the edge of the testing facet as the reference vertex. Then, we generate a vector from the reference vertex to the other vertex, denoted as vector $\mathbf{V}_1$. Another vector, $\mathbf{V}_2$, can be generated from the reference vertex to the testing point. The cosine of the angle θ, formed by the two vectors, can be computed from

$$\cos(\theta) = \frac{\mathbf{V}_1 \bullet \mathbf{V}_2}{|\mathbf{V}_1| \bullet |\mathbf{V}_2|} . \tag{6}$$

The two vectors will coincide if $\cos(\theta) = 1$ and the testing point lies on an edge of the testing facet. However, if a testing point lies very close to but not exactly on an edge of a testing facet, $\cos(\theta)$ will not be exactly equal to one. Therefore, a small tolerance is set to include these cases: points such that $\cos(\theta) > 1 - 2 \times 10^{-4}$ are considered to belong to an edge.

3.4.2 Vertex Points

To check whether or not a testing point lies on a vertex of a testing facet, we calculate the distance between the testing point and the vertex by

$$ds = \sqrt{(v_x - t_x)^2 + (v_y - t_y)^2 + (v_z - t_z)^2} , \tag{7}$$

where $v_{x,y,z}$ are the x, y, and z coordinates of the vertex; $t_{x,y,z}$ are the x, y, and z coordinates of the testing point, and ds is the distance between the testing point and the vertex. A small tolerance needs to be set for the distance comparison of the two points. If the

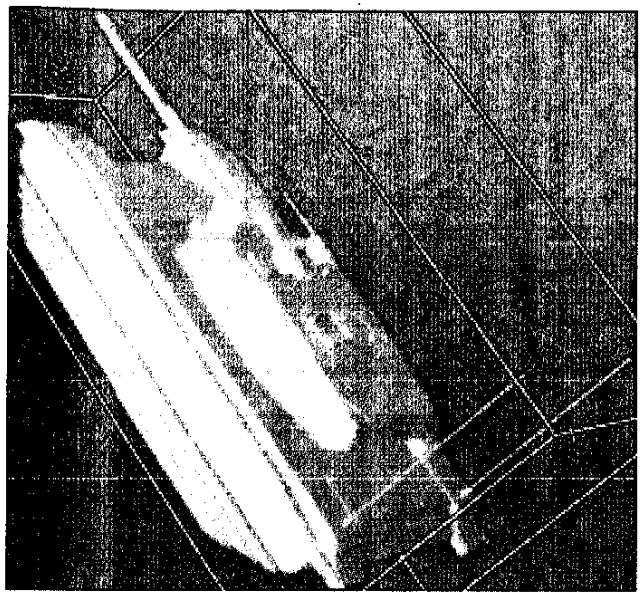

Figure 6. An FDTD grid of a tank.

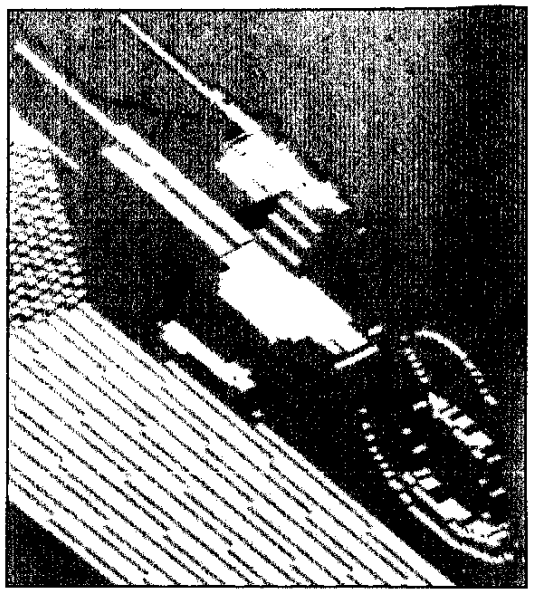

Figure 7. A closer view of the mounted guns in Figure 6.

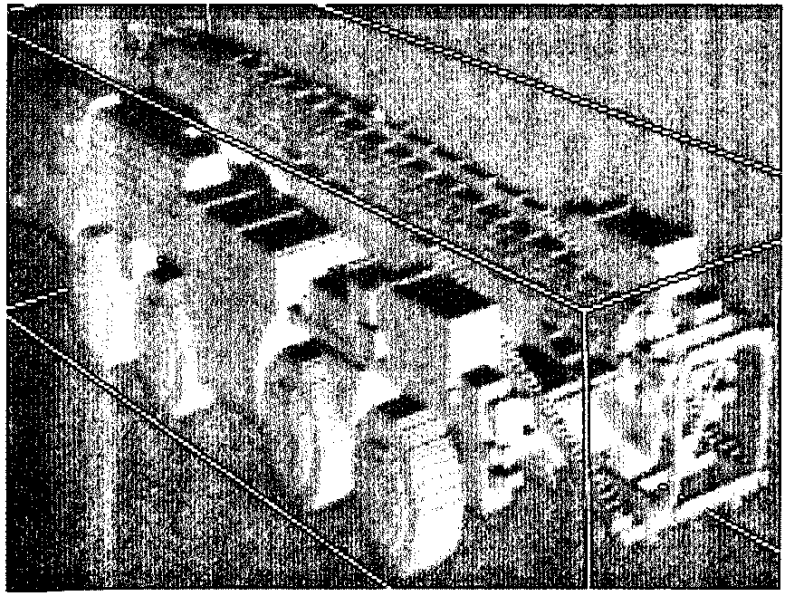

Figure 8a. An FDTD grid of a scud missile and a missile carrier.

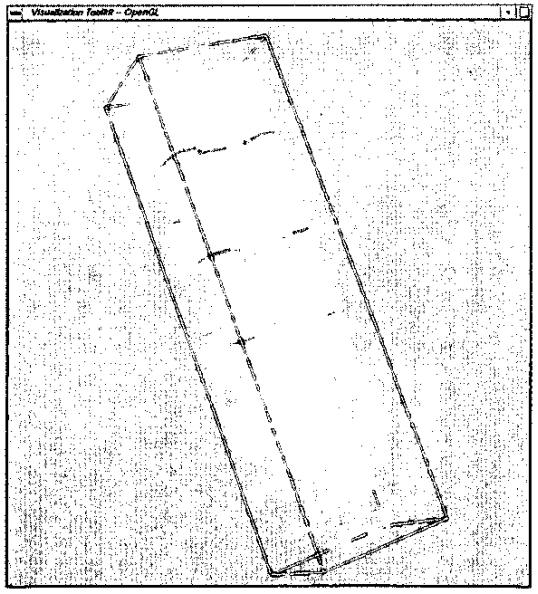

Figure 8b. Another isometric view of an FDTD grid of a scud missile and a missile carrier.

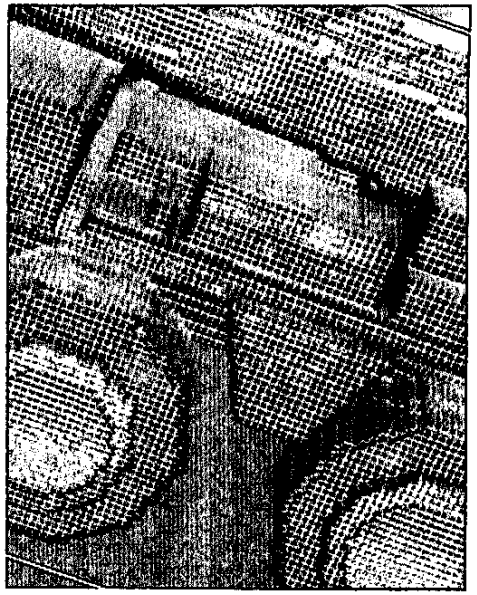

Figure 8c. A close-up look at the scud-missile carrier.

distance *ds* is less than the tolerance, the two points can be considered to be a single point in the grid. The tolerance must be chosen to be much smaller than the (uniform) cell size, in order to ensure that the distance comparison is performed within a uniform cell.

4. Examples

In this section, we illustrate the efficiency and the robustness of this approach for automatically generating three-dimensional FDTD grids. The first example consists of an FDTD grid of an M-1 tank. The CAD description of the tank was represented by 84 triangular facets with 337 nodes, and 227 quadrilateral facets with 1127 nodes. Figure 6 depicts a three-dimensional view of the resulting FDTD grid. The cell size in the FDTD grid was 0.004 m, or 4 cm. The FDTD grid consisted of 1.4 million cells ($86 \times 241 \times 68$ cells). From Figures 6a and 6b, we observe that small details of the tank, such as the mounted machine guns and the two antenna poles, were well captured in the FDTD grid at this resolution. In particular, a closer view of the mounted machine guns is depicted in Figure 7.

Two isometric views of the FDTD grid of a Scud missile launcher, with a Scud missile aboard, are shown in Figures 8a and 8b. The CAD description of the Scud missile launcher and missile, used to generate the FDTD grid, are as follows. The Scud missile launcher and missile were described by 5286 triangular facets with 10573 nodes. The cell size in the FDTD grid for this example was 0.01 m, or 1 cm. The FDTD grid consisted of 202 million cells ($400 \times 1466 \times 344$ cells). In this example, the FDTD grid could have been generated either with the missile and the missile launcher combined as one object, or with the missile and the launcher considered to be two separate objects. Moreover, this figure shows that the algorithm works well for generating the FDTD grid of curved objects, such as illustrated by the wheels of the missile launcher and the scud missile. The fine details of a wheel of the missile launcher are shown in Figure 8c, where the staircase effect inherent in the FDTD method is clearly visible.

5. Conclusions

An approach for automatic three-dimensional FDTD grid generation from CAD files, based on methods used in computer graphics, has been discussed in this paper. This approach results in a robust and efficient algorithm for automatic FDTD grid generation, which eliminates the need for manually creating the FDTD grids for complex objects. Possible future research involves implementation of lower-dimensional objects (surfaces and lines), and their combination with volumetric objects in the FDTD grids. This extension could be useful for applications such as thin-wire approximations of wire antennas mounted on platforms.

6. References

1. A. Taflove and S. Hagness, *Computational Electrodynamics: The Finite-Different Time Domain Method, Second Edition*, Norwood, MA, Artech House, 2000.

2. J. D. Foley, A. van Dam, S. K. Feiner and J. F. Hughes, *Computer Graphics: Principles and Practice, Second Edition in C*, New York, Addison-Wesley, 1997.

3. J. O'Rourke, *Computational Geometry in C, Second Edition*, Cambridge University Press, 1998.

IEEE Virtual Museum Honored

The IEEE Virtual Museum has been selected for inclusion in the National Science Digital Library Report for Math, Engineering, and Technology by the Internet Scout Project. This project reaches in excess of 100,000 people weekly. To see the listing, access http://scout.cs.wisc.edu/nsdl-reports/met/current/.

The Internet Scout Project has published continually since 1994. It covers only the most useful resources, considering the depth of content, the authority of the source, and how well the information is maintained and presented. A team of professional librarians, educators, and content specialists filter hundreds of announcements, looking for the most valuable and authoritative resources available online. Information about the best of what is found is then summarized, organized, and provided to the Internet community in various formats, including via e-mail and the Web. The Internet Scout Project is sponsored by the US National Science Foundation and by Mellon.

The newest exhibit of the IEEE Virtual Museum, "Thomas Edison: A Lifetime of Invention," funded by the Charles Edison Fund, can be viewed at www.ieee-virtual-museum.org. The original development and continued growth of the IEEE Virtual Museum has primarily been provided by the IEEE Foundation, the IEEE Life Members Committee, and the Trustees of the IEEE History Center. Those interested in helping to expand the IEEE Virtual Museum should contact the IEEE Development Office, Tel: +1 (732) 562-3915; E-mail: supportieee@ieee.org. Further information can be obtained from Kim Breitfelder, The IEEE History Center, E-mail: k.breitfelder@ieee.org.

[Information for the above item was taken from an IEEE press release.]