INTERNATIONAL JOURNAL OF NUMERICAL MODELLING : ELECTRONIC NETWORKS, DEVICES AND FIELDS,
Vol. 6, 195–206 (1993)

# EFFICIENT IMPLEMENTATION ISSUES OF FINITE DIFFERENCE TIME-DOMAIN CODES FOR MAXWELL'S EQUATIONS

JOE LoVETRI

*Department of Electrical Engineering, University of Western Ontario, London, Ontario, Canada, N6A 5B9*

AND

GEORGE I. COSTACHE

*Department of Electrical Engineering, University of Ottawa, Ottawa, Ontario, Canada, K1N 6N5*

## SUMMARY

The computer implementation of time-domain finite difference methods for the solution of Maxwell's equations is considered. As the basis of this analysis, Maxwell's equations are expressed as a system of hyperbolic conservation laws. It is shown that, in this form, all the well-known differencing schemes can be easily expressed, thus increasing the applicability of the implementation issues to be discussed. Practical issues, such as computational efficiency and memory requirements, are discussed for the implementation of the finite difference schemes. Advanced programming techniques in the C language are used to implement the finite difference schemes discussed. The example of the penetration of electromagnetic energy through a shield with a thick gap is used to check the performance of the methods. It is shown that, for cases where the disturbance remains localized in the computational mesh, these techniques result in memory and CPU time savings.

## 1. INTRODUCTION

The physical phenomenon of electromagnetic interactions is described by Maxwell's equations. These equations can be expressed in a variety of mathematical forms, each having its own advantages and disadvantages. They may be formulated as a system of partial differential equations over the space and time domain or, as is often done in the engineering literature, the partial differential equations may be immediately cast in the frequency domain by assuming time harmonic excitations.[1,2] Time-domain responses are then calculated by a linear superposition of the individual harmonic solutions. Other formulations making use of integral relations, in either the time or the frequency domains, for either potential fields or the electric and magnetic fields can also be used.

Once the equations have been formulated, many algorithmic techniques exist to produce the required solution. *Finite difference* and *finite element* methods (i.e. including the *method of moments*) form the majority of these techniques for all the different types of formulations encountered.[3–5] Depending on the chosen formulation each algorithmic method has its own advantages and disadvantages. In general, no one algorithm is robust enough to account for all the physical and formulation variations encountered in solving electromagnetic field problems. In recent years much attention has been given to the finite difference approximation of the full time-domain Maxwell's equations.

Practical issues such as memory requirements, efficiency, and overall speed of calculation are of concern when numerical methods are implemented on a computer. To allow easy study of electromagnetic phenomena, the output from a numerical method must also be presented in some visual form (i.e. using graphics software). A summary of all these concerns is depicted in Figure 1.

In the following paper, an investigation is made of improving the computer calculation performance of finite difference techniques based on the differential form of the time-domain Maxwell's equations. Time-domain finite difference techniques based on Maxwell's equations written in

0894–3370/93/030195–12$11.00
© 1993 by John Wiley & Sons, Ltd.
*Received 14 April 1992*
*Revised 4 September 1992*

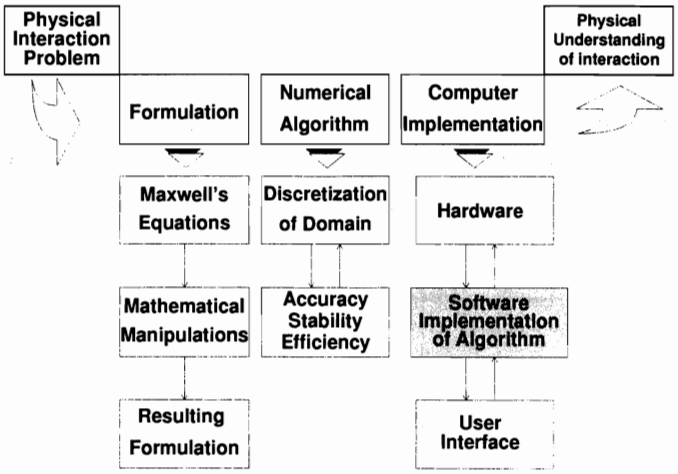



Figure 1. Numerical methods issues

*conservation law* form are implemented. The code is written in the C language and runs on a Sun workstation. Results are shown for the case of an impulsive plane-wave electromagnetic field incident on a two-dimensional shield. This results in a sparse solution which yields to efficient coding methods.

### 1.1. *Formulation of Maxwell's equations in conservation law form*

It can be easily shown (References 5, pp. 180–181, 6 and 7) that Maxwell's and the constitutive relations can be written in conservation form as

$$u_t^j + \frac{\partial E^j}{\partial x} + \frac{\partial F^j}{\partial y} + \frac{\partial G^j}{\partial z} = S^j, \qquad j = 1, \ldots, n \tag{1}$$

where the solution vector $\{u\}$ is given by

$$\{u(\mathbf{x}, t)\} = \left[\begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} \begin{bmatrix} D_x \\ D_y \\ D_z \end{bmatrix}\right]^{\mathrm{T}} = \begin{bmatrix} \mathbf{B} \\ \mathbf{D} \end{bmatrix} \tag{2}$$

and the flux vectors $\{E\}$, $\{F\}$, and $\{G\}$ are given as

$$\{E(\mathbf{x}, t)\} = \left[\begin{bmatrix} 0 \\ \left(\frac{-D_z}{\varepsilon}\right) \\ \left(\frac{D_y}{\varepsilon}\right) \end{bmatrix} \begin{bmatrix} 0 \\ \left(\frac{B_z}{\mu}\right) \\ \left(\frac{-B_y}{\mu}\right) \end{bmatrix}\right]^{\mathrm{T}} = \begin{bmatrix} \hat{x} \times \mathbf{E} \\ -\hat{x} \times \mathbf{H} \end{bmatrix} = \begin{bmatrix} \hat{x} \times e\mathbf{D} \\ -\hat{x} \times m\mathbf{B} \end{bmatrix} \tag{3}$$

$$\{F(\mathbf{x}, t)\} = \left[\begin{bmatrix} \left(\frac{D_z}{\varepsilon}\right) \\ 0 \\ \left(\frac{-D_x}{\varepsilon}\right) \end{bmatrix} \begin{bmatrix} \left(\frac{-B_z}{\mu}\right) \\ 0 \\ \left(\frac{B_x}{\mu}\right) \end{bmatrix}\right]^{\mathrm{T}} = \begin{bmatrix} \hat{y} \times \mathbf{E} \\ -\hat{y} \times \mathbf{H} \end{bmatrix} = \begin{bmatrix} \hat{y} \times e\mathbf{D} \\ -\hat{y} \times m\mathbf{B} \end{bmatrix} \tag{4}$$

$$\{G(\mathbf{x},t)\} = \begin{bmatrix} \begin{bmatrix} \left(\frac{-D_y}{\varepsilon}\right) \\ \left(\frac{D_x}{\varepsilon}\right) \\ 0 \end{bmatrix} & \begin{bmatrix} \left(\frac{B_y}{\mu}\right) \\ \left(\frac{-B_x}{\mu}\right) \\ 0 \end{bmatrix} \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} \hat{z} \times \mathbf{E} \\ -\hat{z} \times \mathbf{H} \end{bmatrix} = \begin{bmatrix} \hat{z} \times e\mathbf{D} \\ -\hat{z} \times m\mathbf{B} \end{bmatrix} \tag{5}$$

The source term $S^j(\mathbf{x}, t)$ in equation (1) represents the current density in the region of interest and is written as

$$\{S(\mathbf{x},t)\} = \begin{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} & \begin{bmatrix} -J_x \\ -J_y \\ -J_z \end{bmatrix} \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 0 \\ -\mathbf{J} \end{bmatrix} \tag{6}$$

It can be easily checked that equation (5) represents Maxwell's curl equations. For linear regions the divergence relations will be automatically satisfied as long as they are satisfied by the initial conditions.

The above formulation makes it is easier to talk about finite difference methods for Maxwell's equations. In fact, in the next section it will be shown how the popular Yee algorithm[8,9] and others can be represented for the above system. This will allow the formalization of sparse calculation techniques for most of the finite difference methods in the literature.

## 2. REPRESENTATION OF FINITE DIFFERENCE SCHEMES

The numerical solution of initial boundary value problems using finite difference techniques is now discussed. The convenience of describing Maxwell's equations in conservation law form will become evident as standard finite difference techniques are described for systems of conservation laws in three space dimensions. Reference to more accurate *upwind* schemes which use numerical fluxes defined via the method of characteristics are also given.

An approximate solution to the general three-dimensional conservation law,

$$\mathbf{u}_t + \mathbf{E}_x + \mathbf{F}_y + \mathbf{G}_z = \mathbf{S} \tag{7}$$

with initial conditions $\mathbf{u}(x,y,z,0) = \mathbf{g}(x,y,z)$ is sought on the rectangular lattice

$$\Omega_{jkln} = \left\{ (x,y,z,t) \,\middle|\, \begin{bmatrix} x_{j-1/2} \leqslant x \leqslant x_{j+1/2} \\ y_{k-1/2} \leqslant y \leqslant y_{k+1/2} \\ z_{l-1/2} \leqslant z \leqslant z_{l+1/2} \\ t^n \leqslant t \leqslant t^{n+1} \end{bmatrix} \right\} \tag{8}$$

where $x_j = j\Delta x$, $y_k = k\Delta y$, $z_l = l\Delta z$, $t^n = n\Delta t$ and $j$, $k$, $l$, and $n$ are integers. The initial conditions are approximated on $\Omega_{jkl0}$ by

$$\mathbf{u}^0_{jkl} = \mathbf{g}_{jkl} = \mathbf{g}(x_j, y_k, z_l) \tag{9}$$

and a finite difference procedure is used to determine the solution at a new time $t = t^n$. A general explicit finite difference scheme can be written as

$$\mathbf{u}^{n+1}_{jkl} = Q\,(\mathbf{u}^n_{jkl}) \tag{10}$$

for $n \geqslant 0$ where $Q$ is a polynomial in the backward and forward shift operators $\mathsf{E}_-$ and $\mathsf{E}_+$ for each space dimension. The qualities of *consistency, stability*, and *convergence* are used to describe specific difference scheme.[10,11]

### 2.1. *Leap-frog: Yee algorithm*

The simplest scheme to understand and implement is the leap-frog scheme formulated by Yee.[8,9] This is a two-step scheme on an interlaced mesh. The interlacing of the solution vector components is easy to see in one dimension, but it is a bit more difficult to visualize in two and three dimensions. It is important to note that this interlacing is specific to Maxwell's equations and may not occur for other equations.

The solution vector may be approximated as

$$\mathbf{u}^n_{jkl} = \begin{bmatrix} \mathbf{B}^n_{jkl} \\ \mathbf{D}^n_{jkl} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} (B_x)^{n-1/2}_{j,k+1/2,l+1/2} \\ (B_y)^{n-1/2}_{j+1/2,k,l+1/2} \\ (B_z)^{n-1/2}_{j+1/2,k+1/2,l} \end{bmatrix} & \begin{bmatrix} (D_x)^n_{j+1/2,k,l} \\ (D_y)^n_{j,k+1/2,l} \\ (D_z)^n_{j,k,l+1/2} \end{bmatrix} \end{bmatrix}^{\mathrm{T}} \tag{11}$$

and the scheme is written as

$$\begin{aligned} \mathbf{D}^{n+1}_{jkl} = \mathbf{D}^n_{jkl} &+ \Delta t(\mathbf{s}^{n+1}_{jkl} + \mathbf{s}^n_{jkl}) \\ &- \rho_x(\mathbf{E}^n_j - \mathbf{E}^n_{j-1}) \\ &- \rho_y(\mathbf{F}^n_k - \mathbf{F}^n_{k-1}) \\ &- \rho_z(\mathbf{G}^n_l - \mathbf{G}^n_{l-1}) \end{aligned} \tag{12}$$

and

$$\begin{aligned} \mathbf{B}^{n+1}_{jkl} = \mathbf{B}^n_{jkl} &+ \Delta t(\mathbf{s}^{n+1}_{jkl} + \mathbf{s}^n_{jkl}) \\ &- \rho_x(\mathbf{E}^n_{j+1} - \mathbf{E}^n_j) \\ &- \rho_y(\mathbf{F}^n_{k+1} - \mathbf{F}^n_k) \\ &- \rho_z(\mathbf{G}^n_{l+1} - \mathbf{G}^n_l) \end{aligned} \tag{13}$$

where $\mathbf{B}^n_{jkl}$ implies $\mathbf{u}^n_{jkl}$ for any of the $\mathbf{B}$ components and $\mathbf{D}^n_{jkl}$ implies $\mathbf{u}^n_{jkl}$ for any of the $\mathbf{D}$ components and $\rho_\xi = \Delta\xi/\Delta t$. The notation for the flux vectors $\mathbf{E}$, $\mathbf{F}$, and $\mathbf{G}$ is determined by

$$\mathbf{E}^n_j = \mathbf{E}(\mathbf{u}^n_{jkl}) = \begin{bmatrix} \begin{bmatrix} 0 \\ (-eD_z)^n_{j,k,l+1/2} \\ (eD_y)^n_{j,k+1/2,l} \end{bmatrix} & \begin{bmatrix} 0 \\ (mB_z)^{n-1/2}_{j+1/2,k+1/2,l} \\ (-mB_y)^{n-1/2}_{j+1/2,k,l+1/2} \end{bmatrix} \end{bmatrix}^{\mathrm{T}} \tag{14}$$

$$\mathbf{F}^n_k = \mathbf{F}(\mathbf{u}^n_{jkl}) = \begin{bmatrix} \begin{bmatrix} (eD_z)^n_{j,k,l+1/2} \\ 0 \\ (-eD_x)^n_{j+1/2,k,l} \end{bmatrix} & \begin{bmatrix} (-mB_z)^{n-1/2}_{j+1/2,k+1/2,l} \\ 0 \\ (mB_x)^{n-1/2}_{j+k+1/2,l+1/2} \end{bmatrix} \end{bmatrix}^{\mathrm{T}} \tag{15}$$

$$\mathbf{G}^n_l = \mathbf{G}(\mathbf{u}^n_{jkl}) = \begin{bmatrix} \begin{bmatrix} (-eD_y)^n_{j,k+1/2,l} \\ (eD_x)^n_{j+1/2,k,l} \\ 0 \end{bmatrix} & \begin{bmatrix} (mB_y)^{n-1/2}_{j+1/2,k,l+1/2} \\ (-mB_x)^{n-1/2}_{j,k+1/2,l+1/2} \\ 0 \end{bmatrix} \end{bmatrix}^{\mathrm{T}} \tag{16}$$

where only the rows of the flux vectors which are required are used in equations (12) and (13). The component locations as defined by equation (11) are depicted in Figure 2.

In two dimensions the scheme is obvious and will not be given explicitly here, except to recall that Maxwell's equations uncouple into two independent sets of three partial differential equations. These are the *TE case* where only the $B_z$, $D_x$, and $D_y$ components exist, and the *TM case* where only the $D_z$, $B_x$, and $B_y$ components exist. Note that these two cases correspond to either the components at the bottoms of the cubes in Figure 2 (i.e. TE case) or at the tops of the cubes (i.e. TM case).

In order to start the time marching, when either the initial $\mathbf{B}$ or the initial $\mathbf{D}$ is given in the

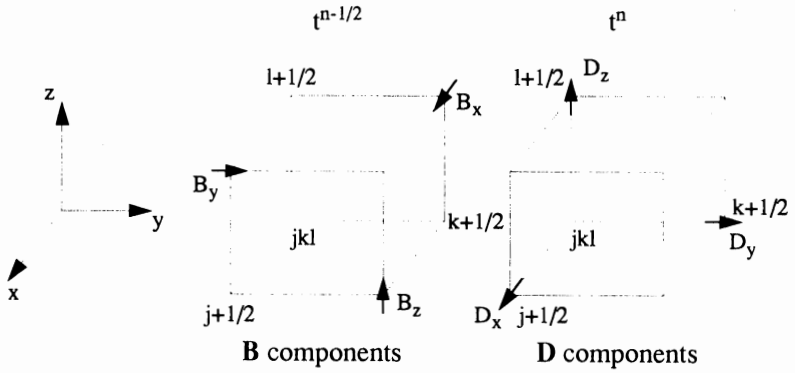



Figure 2. Location of components for leap-frog scheme

initial condition, the first step will be a half timestep ($\Delta t^0 = \Delta t/2$) if the other component is assumed to be zero. For example, if $\mathbf{B}(x, 0)$ is given as the initial condition, then equations (12) are used to update the **D** fields assuming that the previous **D** fields are zero, and using a timestep of $\Delta t/2$. Alternatively, if $\mathbf{D}(x, 0)$ is given then equations (13) are used first with a timestep of $\Delta t/2$. From this point on equations (13) and equations (12) are used alternately with a $\Delta t$ timestep.

The boundary conditions for a perfectly conducting medium are easily implemented by imposing that the tangential electric field be equal to zero on this boundary. Thus, when a perfectly conducting body is placed in the mesh, the boundaries of the body should line up with mesh points containing tangential **E** field components.

The stability of the scheme has been established by many authors.[9] The scheme is stable for

$$\Delta t \leqslant \frac{1}{\left(\dfrac{1}{\Delta x^2} + \dfrac{1}{\Delta y^2} + \dfrac{1}{\Delta z^2}\right)^{1/2} c_{\max}} \tag{17}$$

where if $\Delta x = \Delta y = \Delta z$ this corresponds to a Courant number of $1/(\sqrt{3})$. In two dimensions, $\Delta z = \infty$ and the Courant number becomes $1/(\sqrt{2})$.

## 2.2. *Lax-Wendroff: two-step method*

There are many ways to write the Lax-Wendroff scheme.[4] The formulation of the Lax-Wendroff method in two-step form was first presented by Richtmyer[12] and thus the different versions of the scheme have become his namesake. The analysis of these schemes has been investigated by Wilson[13] and are written below where $\mu$ is the averaging operator $\mu_x u_j = (1/2)\,(u_{j+1/2} + u_{j-1/2})$ and $\delta$ is the central difference operator $\mu_x u_j = (1/2)\,(u_{j+1/2} - u_{j-1/2})$.

*(i) Richtmyer scheme (also two-step Lax-Wendroff)*

$$\mathbf{u}^{n*}_{jkl} = \frac{(\mu_x + \mu_y + \mu_z)}{3}\,\mathbf{u}^{n}_{jkl} - \frac{[\rho_x\delta_x\mathbf{E}^{n}_{jkl} + \rho_y\delta_y\mathbf{F}^{n}_{jkl} + \rho_z\delta_z\mathbf{G}^{n}_{jkl}]}{2}$$

$$\mathbf{u}^{n+1}_{jkl} = \mathbf{u}^{n*}_{jkl} - [\rho_x\delta_x\mathbf{E}^{n*}_{jkl} + \rho_y\delta_y\mathbf{F}^{n*}_{jkl} + \rho_z\delta_z\mathbf{G}^{n*}_{jkl}] \tag{18}$$

*(ii) Modified Richtmyer scheme*

$$\mathbf{u}^{n*}_{jkl} = (\mu_x\mu_y\mu_z)\,\mathbf{u}^{n}_{jkl} - \frac{[\rho_x\delta_x\mathbf{E}^{n}_{jkl} + \rho_y\delta_y\mathbf{F}^{n}_{jkl} + \rho_z\delta_z\mathbf{G}^{n}_{jkl}]}{2}$$

$$\mathbf{u}^{n+1}_{jkl} = \mathbf{u}^{n*}_{jkl} - [\rho_x\delta_x\mathbf{E}^{n*}_{jkl} + \rho_y\delta_y\mathbf{F}^{n*}_{jkl} + \rho_z\delta_z\mathbf{G}^{n*}_{jkl}] \tag{19}$$

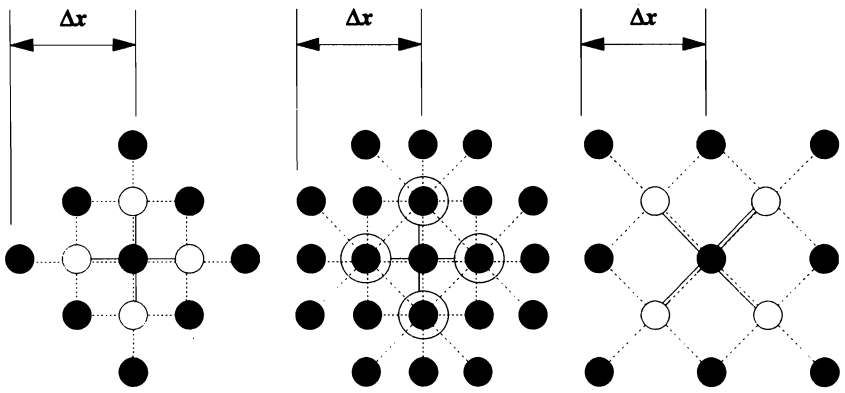



Figure 3. Computational molecules for 2-D Richtmyer schemes

*(iii) Rotated Richtmyer scheme*

$$\mathbf{u}_{jkl}^{n*} = (\mu_x\mu_y\mu_z)\,\mathbf{u}_{jkl}^{n} - \frac{[\rho_x\mu_y\mu_z\delta_x\mathbf{E}_{jkl}^{n} + \rho_y\mu_x\mu_z\delta_y\mathbf{F}_{jkl}^{n} + \rho_z\mu_x\mu_y\delta_z\mathbf{G}_{jkl}^{n}]}{2}$$

$$\mathbf{u}_{jkl}^{n+1} = \mathbf{u}_{jkl}^{n*} - [\rho_x\mu_y\mu_z\delta_x\mathbf{E}_{jkl}^{n*} + \rho_y\mu_x\mu_z\delta_y\mathbf{F}_{jkl}^{n*} + \rho_z\mu_x\mu_y\delta_z\mathbf{G}_{jkl}^{n*}] \tag{20}$$

In the above schemes the $x$-direction flux vector is approximated as $\mathbf{E}_{jkl}^{n} = \mathbf{E}(\mathbf{u}_{jkl}^{n})$, $\mathbf{E}_{jkl}^{n*} = \mathbf{E}(\mathbf{u}_{jkl}^{n*})$, with similar relations for the flux vectors $\mathbf{F}$, and $\mathbf{G}$, and $\rho\xi = \Delta\xi/\Delta t$. The two-dimensional versions of these schemes can be easily obtained from the above equations by dropping the flux vector $\mathbf{G}$ and all finite difference operators in the $z$-co-ordinate direction. The two-dimensional computational molecules for each of these three schemes are shown in Figure 3.

Note that in these schemes every component of the solution vector $\mathbf{u}$ is represented at each lattice point. Thus it is simpler to introduce material boundaries into the mesh. The rotated Richtmyer scheme is the simplest on which to impose boundary conditions owing to its square computational boundary. It also has the largest Courant number. The Courant numbers for the Richtmyer schemes,[13] corresponding to the case where $\Delta x = \Delta y = \Delta z$, are given in Figure 4.

## 2.3. *Upwind higher-order methods*

Other finite difference techniques known as upwind methods[14–17] which make use of higher order flux representations[18–20] can also be formulated for Maxwell's equations in conservation form.[6,7] The details of these schemes will not be given here but a practical comparison of these as well as the abovementioned schemes can be found in LoVetri *et al.*[21]

| *Scheme* | *2-D Courant No.* | *3-D Courant No.* |
|---|---|---|
| Richtmyer | $\frac{1}{\sqrt{2}}$ | $\frac{1}{\sqrt{3}}$ |
| Modified Richtmyer | $\frac{1}{\sqrt{2}}$ | $\frac{1}{\sqrt{3}}$ |
| Rotated Richtmyer | 1 | 1 |

Figure 4. Courant numbers for Richtmyer schemes

## 3. PRACTICAL IMPLEMENTATION AND RESULTS

The finite difference schemes presented above were implemented for the two-dimensional case on a Sun Microsystems' SPARCstation 1™. In order to achieve the greatest utility from the program, special attention was given to the practical aspects of the implementation. That is, *computational speed* and *memory requirements*. The main difficulty lies not with the speed of the computations, since all the schemes discussed are explicit schemes, but rather with the storage space required for the schemes.

### 3.1. *Memory requirements*

In three-dimensional space, the memory requirements for a practical-sized problem can be formidable. For example, the total memory requirement in bytes, $B_T$, for a finite difference program can be estimated as

$$B_T = N_x \times N_y \times N_z \times B_{node} = N_T \times B_{node} \tag{21}$$

where $N_x$, $N_y$, and $N_z$ are the number of mesh points in the $x$-, $y$-, and $z$-directions respectively (assuming a rectangular lattice), and $B_{node}$ is the number of bytes of storage space required at each node in the lattice. If a scheme is used where all six components of **D** and **B** are stored at every node, then

$$B_{node} \geqslant 6 \times B_{component} \tag{22}$$

where $B_{component}$ is the number of bytes required for each component of the solution vector. If *double precision* is used then $B_{component} = 8$ (for the SPARCstation 1). As well as these, the conductivity, permittivity, and permeability associated with each nodal cell may have to be stored, especially for heterogeneous structures. Thus, the number of bytes per node becomes

$$B_{node} \geqslant 6 \times B_{component} + 3 \times B_{material} \tag{23}$$

where $B_{material}$ corresponds to the bytes required to store the material constants. If *single precision* is used for these, then $B_{material} = 4$ (again for the SPARCstation 1) and $B_{node}$ is at least equal to 60. That is, 60 bytes of storage space for each node in the lattice. For a lattice with $N_x = N_y = N_z = 100$, the total storage requirement for the problem, given by equation (23), is 60 Mbytes. This does not include *overhead* storage which will be required, such as for the storage of the program itself and other intermediate variables used in the calculation. A lattice of this size is not, by any means, large enough to accurately represent a practical electromagnetic interaction problem such as the coupling of energy to a communication system. On the other hand the 60 Mbytes of memory required for this problem is well beyond the capabilities of most engineering workstations (for example the colour SPARCstation 1 being used has a maximum of 16 Mbytes of memory available).

The above memory requirements assume brute force implementation of the schemes. Using state-of-the-art programming techniques, these requirements can be modestly reduced, but not to the extent required to solve electromagnetic interaction problems in complex systems. Thus, these schemes are currently feasible only for the study of specific interaction mechanisms such as coupling through shields and onto cables, or possibly, with the use of a supercomputer, for such problems as the penetration of electromagnetic energy into biological tissues.[9]

### 3.2. *Practical implementation: dynamic memory allocation*

In order to take advantage of advanced programming techniques, the C language has been used in the implementation of the finite difference schemes. This allows the dynamic allocation of storage space to the lattice nodes only when it is required. During the solution of a typical electromagnetic interaction problem it is often the case that many of the nodes at a certain timestep do not contain a disturbance. This is especially true when the excitation is in the form

of a sharp time-domain pulse. For example, in the propagation of an electromagnetic pulse through a shield, the disturbance is localized in space at early interaction times and only starts to spread spatially as the interaction progresses.

This spatial localization of the disturbance can be taken advantage of when a scheme is implemented by allocating memory to a lattice node only if it contains a disturbance at the current timestep *or* has the possibility of containing a disturbance at the next timestep. In the same way, memory is *unallocated* or removed from a lattice node when the node does not contain a disturbance at the current timestep *and* does not have the possibility of containing a disturbance at the next timestep. This allocating and unallocating of memory has the effect of *propagating* the memory storage for the nodes through the lattice with the disturbance.

This process is implemented by first only allocating memory to an array of *pointers* to the data structures which hold the information for each lattice node. Thus, if 10,000 nodes make up a $100 \times 100$ finite difference lattice, then a $100 \times 100$ array of pointers to the nodal data structure is allocated, requiring a total of 4 Kbytes of memory (a pointer requires 4 bytes of memory). The nodal data structure consists of the data types required to represent the solution vector at each node and to keep the material constants for the node's cell. For the three-dimensional Maxwell's equations each data structure consists of six double precision variables. Since the material inhomogeneities are usually constant with time, it is best to define a separate data structure for the three single precision variables required to hold the material constants for the lattice. These data structures are illustrated in Figure 5.

The next step is to input the initial conditions into the lattice as well as the required material constants. This consists of allocating the data structure memory for each node required by the initial conditions to have a non-zero disturbance at time zero. The actual value of the disturbance is then stored in the appropriate data location for a given solution vector component. If a cell contains material constants which are not that of free space, then that cell's material constant data structure memory is allocated and the specific material constants for that cell are stored. In this way, only cells with non-free space material constants have their material constant data structure memory allocated.

Once the initial conditions are input, the finite difference algorithm is applied to find the disturbance at the next timestep. This consists of performing three steps:

(1) allocate nodes which are neighbours to a non-zero node,
(2) execute the finite difference algorithm, and
(3) unallocate nodes which are zero *and* have all neighbours as zero.

In this context, the term neighbour is dependent on the domain of dependence (or the spatial bandwidth) of the finite difference scheme. For example, in the leap-frog and Lax-Wendroff schemes, a disturbance at one node can propagate only to the nearest neighbours of that node in each spatial direction. On the other hand, in the upwind scheme, a disturbance at one node can propagate to its *two* nearest neighbours in each spatial direction.

The above procedure helps in the problem of large memory requirements. In order to save on the time required to compute the disturbance at each timestep, an obvious technique is to not

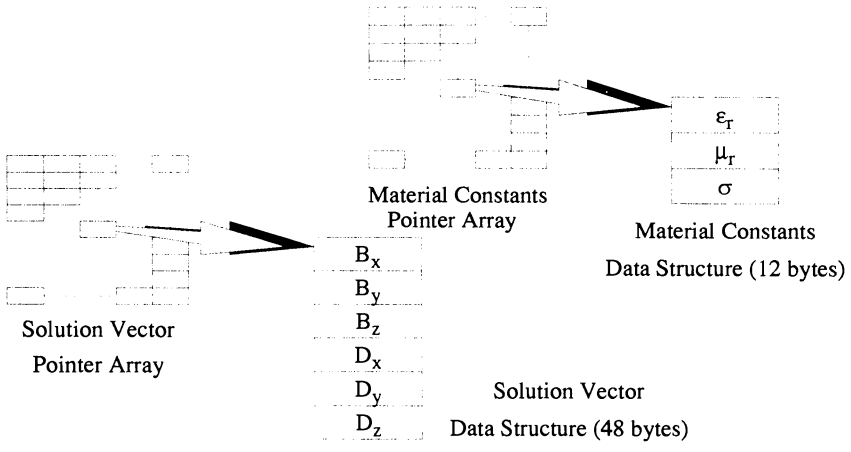



Figure 5. Example of solution vector and material constants data structures

perform the finite difference algorithm at unallocated nodes. If the solution vector data structure of a node is unallocated then it has already been determined *a priori* that a disturbance cannot propagate to this node at the next timestep, thus it is justified to skip the calculation altogether.

In order that every node in the lattice does not have to be checked as to whether it is *in* (i.e.

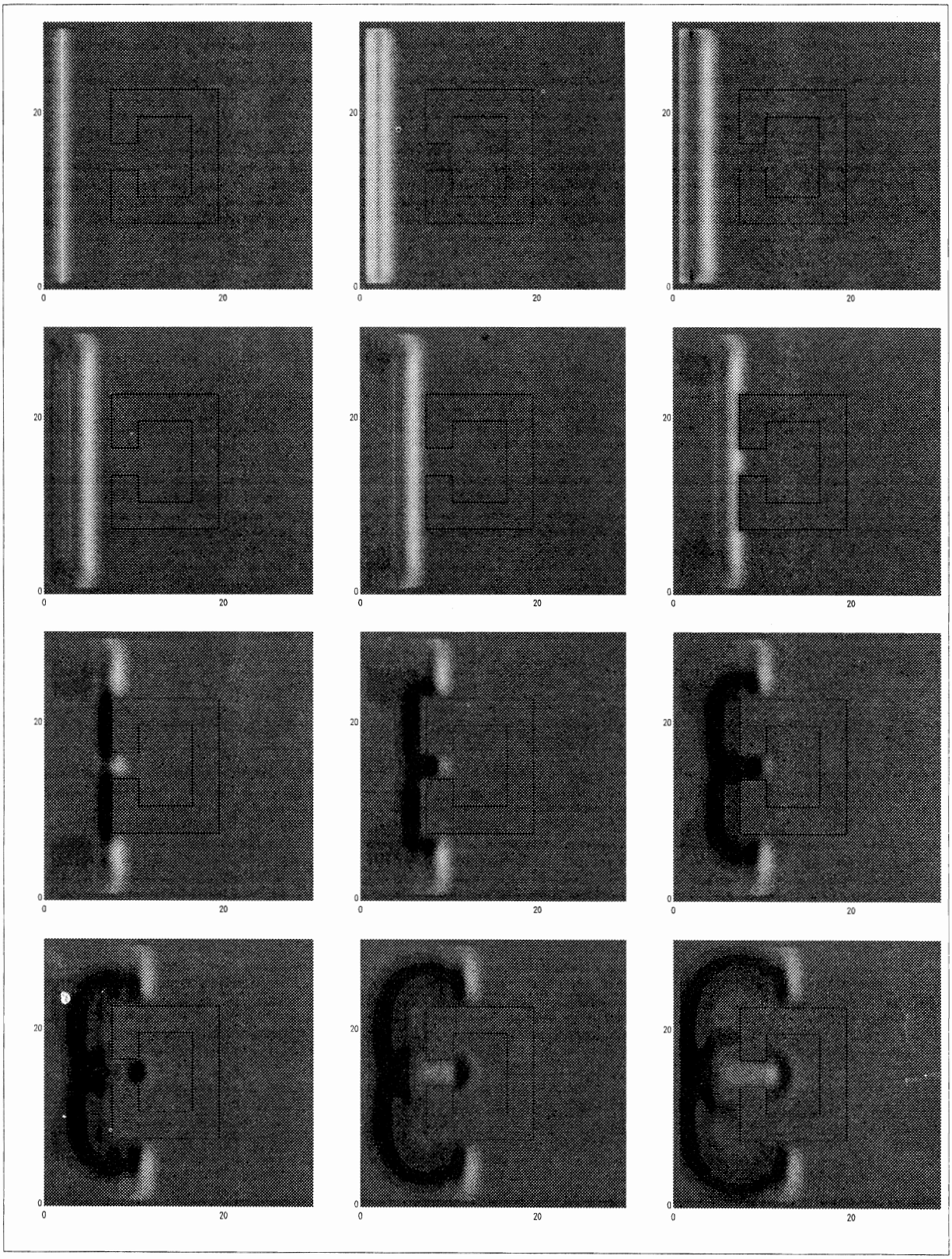



Figure 6. Time sequence of contour plots for the test problem

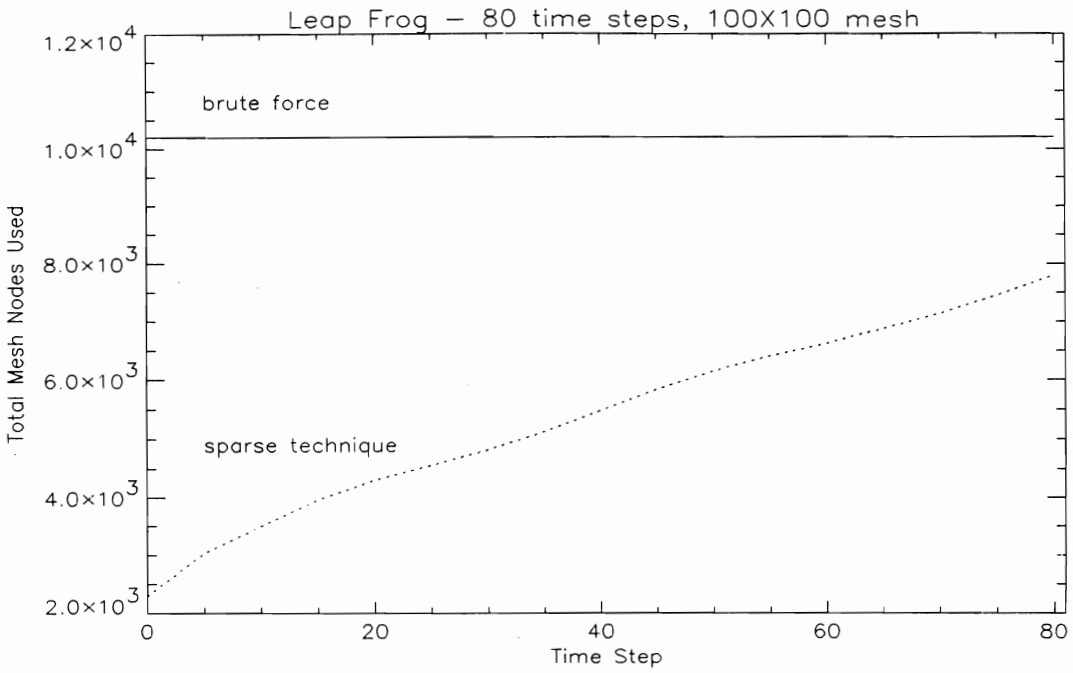



Figure 7. Memory requirement (mesh nodes allocated)

allocated) or *out* (i.e. unallocated) before a calculation is performed, all the nodes which are *in* are kept in a *linked list.*[22] The finite difference algorithm is then only performed on the nodes in the linked list. For example, at time zero only the nodes with given initial conditions along with their neighbours are in the linked list. In this way the calculations at each timestep are performed quicker when there are only a few nodes *in* the lattice. This is usually the case at the beginning of a problem when the disturbance is fairly localized. Every time a discontinuity is met in the lattice the disturbance is *spread* spatially and more nodes will be *in*, having the effect of slowing the computation time for each timestep.

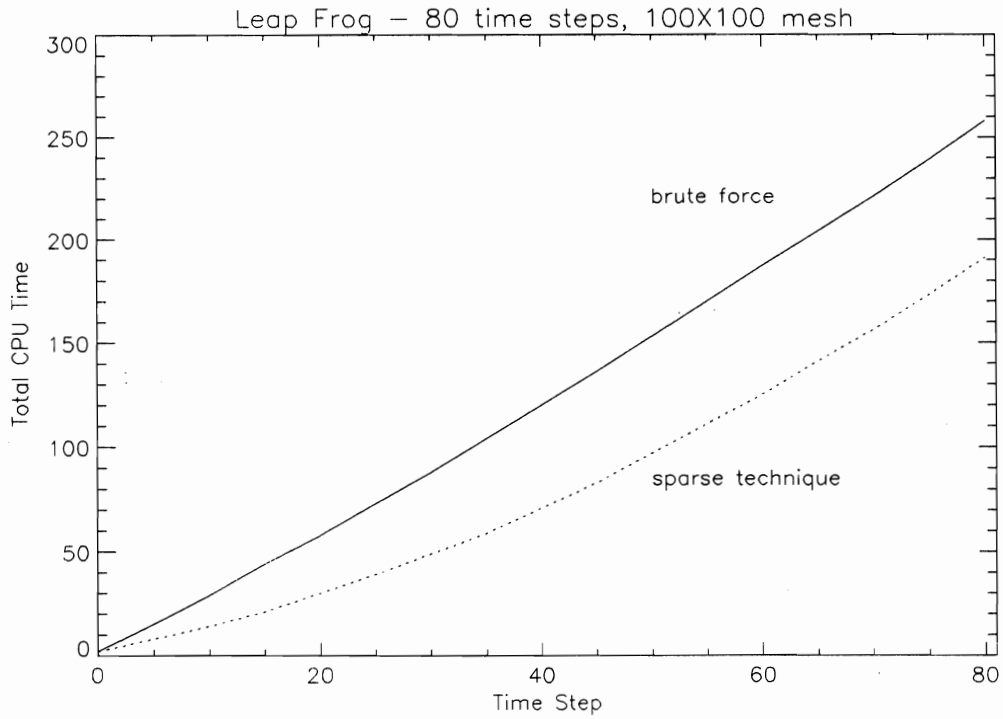



Figure 8. CPU time (s) comparisons

### 3.3. *Performance results*

The two-dimensional leap-frog method was run over 80 timesteps on a 100 × 100 node mesh for the problem of a sharp Gaussian pulse impinging on a thick shield with a gap. A time sequence of contour plots for this problem is shown in Figure 6. The thick conducting shield is shown in the figure and the $x$- and $y$-coordinate axes are given in units of metres. The calculations for this figure are produced via the leap-frog method. Note that high-order absorbing boundary conditions have not been used for this method although many formulations for these are available in the literature.

Results of the memory required at each timestep along with the total CPU time required are shown below in Figures 7 and 8 for both the brute force method and the dynamic allocation (sparse technique). As the pulse fills the inside of the shield, the number of allocated nodes increases and the slope of the 'total CPU time' graph increases. In a sense, the memory allocation for the grid points follows the electromagnetic disturbance. Note that in Figure 8 the brute force method does not produce an exact linear CPU time-dependence with the computational timestep because the time required in writing the output file increases as more non-zero nodes are created. When the writing of the output file was eliminated a linear relationship between computational timestep and total CPU time resulted.

## 4. CONCLUSIONS

Finite difference techniques for solving Maxwell's equations in conservation law form have been discussed and implemented. It has been shown how advanced programming techniques, using dynamic memory allocation to optimize the use of memory during calculations, can be used to efficiently implement all of the discussed finite difference schemes. These sparse excitation techniques can be effectively used for problems with spatially localized propagating disturbances through a finite difference mesh. As the disturbance begins to fill the mesh (e.g. steady state problems) the method may be less efficient due to the extra overhead involved in implementing the method.

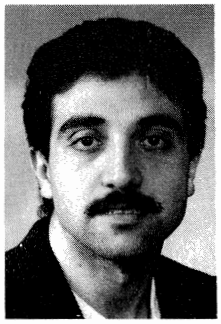
The finite difference method is one of the most powerful methods for the solution of hyperbolic systems of equations. The severe practical limitations in the implementation of three-dimensional finite difference schemes, for use in solving complex interaction problems, have been discussed. These limitations cannot be solved solely by the use of larger and larger computers. Advanced programming methods must be used to implement these algorithms. Also, the study and formulation of more efficient algorithms is an area of research which will have rewarding effects in the field of electromagnetic interaction modelling.

## REFERENCES

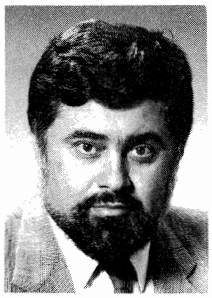
1. R. F. Harrington, *Time-Harmonic Electromagnetic Fields*, McGraw-Hill, New York, 1961.
2. J. A. Stratton, *Electromagnetic Theory*, McGraw-Hill, New York, 1941.
3. M. N. O. Sadiku, *Numerical Techniques in Electromagnetics*, CRC Press, Boca Raton, FL, 1992.
4. L. Lapidus and G. F. Pinder, *Numerical Solution of Partial Differential Equations in Science and Engineering*, John Wiley, New York, 1982.
5. A. R. Mitchell and D. F. Griffiths, *The Finite Difference Method in Partial Differential Equations*, John Wiley, New York, 1980.
6. V. Shankar, W. F. Hall and A. H. Mohammadian, 'A time-domain differential solver for electromagnetic scattering problems', *Proceedings of the IEEE*, **77**, 709–721 (1989).
7. V. Shankar, A. H. Mohammadian and W. F. Hall, 'A time-domain finite-volume treatment for the Maxwell equations', *Electromagnetics*, **10**, 127–145 (1990).
8. S. K. Yee, 'Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media', *IEEE Trans. on Antennas and Propagation*, **AP-14**, 302–307 (1966).
9. A. Taflove and K. R. Umashankar, 'Finite-difference time-domain (FD-TD) modeling of electromagnetic wave scattering and interaction problems', *IEEE Ant. and Prop. Soc. Newsletter*, April, 5–20 (1988).
10. G. A. Sod, *Numerical Methods in Fluid Dynamics: Initial and Initial Boundary-Value Problems*, Cambridge University Press, Cambridge, 1985.
11. J. C. Strikwerda, *Finite Difference Schemes and Partial Differential Equations*, Wadsworth & Brooks/Cole Advanced Books & Software, Pacific Grove, CA, 1989.
12. R. D. Richtmyer and K. W. Morton, *Difference Methods for Initial-Value Problems*, Interscience, New York, 1967.
13. J. C. Wilson, 'Stability of Richtmyer type difference schemes in any finite number of space variables and their comparison with multistep Strang schemes', *J. Inst. Maths. and Applics.*, **10**, 238–257 (1972).
14. A. Harten, P. D. Lax and B. Van Leer, 'On upstream differencing and Godunov-type schemes for hyperbolic conservation laws', *SIAM Review*, **25**, 35–61 (1983).

15. S. Osher and F. Solomon, 'Upwind difference schemes for hyperbolic systems of conservation laws', *Mathematics of Computation*, **38**, 339–374 (1982).
16. R. F. Warming and R. M. Beam, 'Upwind second-order difference schemes and applications in aerodynamic flows', *AIAA Journal*, **14**, 1241–1249 (1976).
17. J. L. Steger and R. F. Warming, 'Flux vector splitting of the inviscid gasdynamic equations with applications to finite-difference methods', *J. of Comp. Phys.*, **40**, 263–293 (1981).
18. S. Osher, 'Riemann solvers, the entropy condition, and difference approximations', *SIAM J. of Numer. Anal.*, **21**, 217–235 (1984).
19. P. L. Roe, 'Approximate Riemann solvers, parameter vectors, and difference schemes', *J. of Comp. Phys.*, **43**, 357–372 (1981).
20. P. K. Sweby, 'High resolution schemes using flux limiters for hyperbolic conservation laws', *SIAM J. of Numer. Anal.*, **21**, 995–1011 (1984).
21. J. LoVetri and M. S. Wartak, 'Practical comparison of finite difference time domain schemes', *1991 IEEE AP-S International Symposium and URSI Radio Science Meeting*, London, Ontario, 24–28 June 1991, pp. 418–421.
22. R. N. Horspool, *C: Programming in the Berkeley UNIX Environment*, Prentice-Hall Canada, Scarborough, Ontario, 1986.

**Authors' biographies:**



**Joe LoVetri** is an Assistant Professor in the Department of Electrical Engineering of the University of Western Ontario in London, Ontario, Canada. He received the B.Sc. and M.Sc. degrees, both in electrical engineering, from the University of Manitoba in 1984 and 1986 respectively. From 1984 to 1986 he was EMI/EMC Engineer at Sperry Defence Division in Winnipeg, Manitoba. From 1986 to 1988 he held the position of TEMPEST engineer at the Communications Security Establishment in Ottawa, Ontario. From 1988 to 1991 he was a Research Officer at the Institute for Information Technology of the National Research Council of Canada. He completed his Ph.D. degree in electrical engineering at the University of Ottawa in January 1991. His main interests lie in the general area of algorithmic and non-algorithmic computer modelling of physical processes, especially electromagnetic interactions.



**George I. Costache** is a Professor in the Electrical Engineering Department at the University of Ottawa, Ottawa, Ontario, Canada. His career has included positions at Bell Northern Research, Ottawa, Ontario, the University of Manitoba and the Polytechnic Institute of Bucharest. He has taught electromagnetics and numerical techniques applied to electromagnetics for more than 26 years and has made original contributions to the solution of skin-effect problems and electromagnetic transient phenomena. His main interest is in numerical techniques, such as finite-element analysis and moment methods, and their application to interference problems in steady-state and time-domain applications.

The author or coauthor of over 50 technical papers and reports, Dr Costache is an Associate Editor of the *IEEE Transactions on Electromagnetic Compatibility*. He is also a member of the editorial review board of COMPEL, the *International Journal for Computation and Mathematics in Electrical and Electronics Engineering, International Journal of Numerical Modelling, Electronic Networks Devices and Fields* and *IEEE Transactions on Microwave Theory and Techniques*. He is a registered Professional Engineer in the province of Ontario, Canada.