High-Order FVTD on Unstructured Grids using an Object-Oriented Computational Engine

Dmitry Firsov[1], Joe LoVetri[1], Ian Jeffrey[1], Vladimir Okhmatovski[1], Colin Gilmore[1], and Walid Chamma[2]

[1]Department of Electrical and Computer Engineering
University of Manitoba, Winnipeg, MB, Canada, R3T 5V6
firsovd@ee.umanitoba.ca, lovetri@ee.umanitoba.ca

[2]Radar Electronic Warfare Section
Defence R&D Canada - Ottawa, Department of National Defence
3701 Carling Avenue, Ottawa, Ontario, K1A 0Z4
Walid.Chamma@drdc-rddc.gc.ca

Abstract—An object-oriented implementation of a finite-volume time-domain (FVTD) engine for solving Maxwell's equations is presented. The relevant aspects of the FVTD method are discussed from an object-oriented perspective and details of the object classes are given. Computational results obtained using the FVTD engine for solving Maxwell's Equations on unstructured grids are also shown. The engine implements both MUSCL and polynomial interpolation methods to approximate the fluxes at the cell boundaries up to third-order accuracy. In addition, the engine has the capability of using a number of time-integration schemes. Results are presented for the transient scattering from a PEC sphere and a lossy dielectric cube. For the case of the sphere, almost perfect agreement with the analytic solution in the time-domain is achieved. The number of cells required as compared to FDTD is substantially reduced.

Keywords—Finite-volume time-domain, FVTD, Maxwell's Equations, object-oriented design.

I. INTRODUCTION

The finite-difference time-domain (FDTD) algorithm is probably the most popular computational electromagnetics (CEM) technique in use today. The two main drawbacks of the *standard* FDTD method are that curved geometries must be approximated by "stair-stepping" the boundaries and that the electromagnetic field components are interlaced in space and time. These drawbacks require that a fine grid be used in order to resolve curved boundaries which increases the required computational resources. There have been several successful investigations on modifying the method for non-rectangular boundaries but these are difficult to implement and use [1].

In recent years, the computational electromagnetic community has taken interest in the finite-volume time-domain (FVTD) algorithm as an alternative or companion to the simple and powerful FDTD algorithm for solving Maxwell's equations [2, 3, 4]. The primary reason for this interest is that the basic formulation of FVTD does not require a structured spatial mesh and so its ability to solve electromagnetic problems involving complex geometries is not constrained by a lack of ability to accurately describe the physical problem.

The finite-volume technique is a standard technique used in Computational Fluid Dynamics (CFD) [5]. One of the first comprehensive implementations of the technique for CEM was reported by Shankar *et al*. in the early 1990's [2]. Shankar's method collocates all the field components at the center of each finite volume and is implemented on structured body-fitted curvilinear grids. It is a characteristic-based FVTD scheme which uses a two-step second-order upwinding scheme. A similar technique has also been presented by Shang [6]. Recently, two other groups have reported achieving excellent results using a characteristic-based FVTD technique [3, 4]. Both use a second-order accurate Monotone Upstream-centered Scheme for Conservation Laws (MUSCL) to interpolate the fluxes at the finite-volume facets.

In this paper we consider the FVTD method on unstructured grids and present the use of a higher-order flux-interpolatory method, developed by Ollivier-Gooch for fluid-dynamics problems [8], for our FVTD computational engine. In addition, as the implementation of our computational engine was undertaken from an object-oriented perspective, this paper demonstrates the flexibility of such an implementation of the FVTD algorithm for handling a multitude of volumetric mesh descriptions, time-integration, and flux-integration approximations. Also, in order to facilitate an in-depth understanding of the method, we provide an outline of the class hierarchy and programming tactics used during implementation. Although our FVTD implementation is in C++, the object-oriented concepts we discuss are not language specific.

This paper is organized as follows: In Section 2, we discuss the object-oriented implementation of a discretized volumetric mesh. Section 3 presents a brief overview of the theory behind the FVTD method as it pertains to Maxwell's equations. In Section 4, we discuss some of the interpolatory techniques available for flux-integration, namely MUSCL and polynomial interpolation schemes. Section 5 overviews the time-integration schemes considered which retain the operator representation of the FVTD update equation as a matrix-vector-product. Finally, Section 6 validates the engine by comparing computational results using FDTD, FVTD with MUSCL interpolation, and FVTD with third-order interpolation against FDTD. The problems of transient scattering from a PEC sphere and from a lossy dielectric cube are considered.

II. Object-Oriented Mesh Representation

The FVTD solution of Maxwell's equations requires a volumetric grid over a specified three-dimensional region of interest. The meshing software we use[1] is capable of producing unstructured meshes comprised of tetrahedrons, hexahedrons, prisms, or pyramids and so our FVTD engine has been designed to function using any of these volumetric elements. Any volumetric mesh consisting of polyhedral elements may be fully described as a collection of elements which are, in turn, described by vertices and facets. Thus the mesh description inherently includes a geometrical hierarchy. Using an object-oriented approach, our FVTD engine implements a volumetric mesh as an object consisting of instances of elements, vertices and facets each implemented as their own separate classes as depicted in Fig. 1. The mesh itself is an instance of the *cMesh* class and contains the geometrical description of the mesh via arrays of volumetric elements and vertices. A *cMesh* object is responsible for accessing the mesh description from file and is additionally responsible for saving the mesh description in alternative formats compatible with various visualization tools[2]. A brief discussion of the vertex, element and facet descriptions follows.

Each vertex in the mesh, represented by an instance of the *cPoint* class, contains three critical pieces of information: its spatial location, a unique identification tag corresponding to its location in the array of points in the *cMesh* object, and a list of pointers to all elements sharing it. The *cPoint* class also doubles as a general Euclidean vector class and is equipped with standard vector operators such as the cross-product.

Each element in the mesh, represented by an instance of the *cElement* class, is also given a unique identification tag and contains a list of pointers to *cPoint* objects (denoting the vertices of the element) as well as a list of the neighbouring elements. Storing neighbouring elements by their identification tag is essential for an efficient implementation. The element type is specified by a member in the *cElement* class. In addition, the *cElement* class is equipped with a set of utility functions used to compute various geometrical properties of a given instance of the class. The functions include the computation of the element volume and dynamic instantiation of element facets, as they are required throughout the FVTD algorithm, by means of the *cFacet* class. The different types of volumetric elements each require different functions for appropriately computing their geometrical properties. Because simple polyhedral computations can be easily coded inline, our implementation does not exploit inheritance to derive a specific element from a base element class. If, however, higher order elements were of interest (i.e., elements with curved boundaries) such inheritance would be beneficial for more complicated geometrical computations.

Although it is possible to pre-compute and store facet information for each element in the mesh, experience has shown that such a list significantly increases memory requirements. Therefore, as needed, element facets are generated via function calls in the *cElement* class which dynamically instantiate an instance of the *cFacet* class. A facet object is responsible for computing both the area and outward normal of the facet which it stores for use during the FVTD algorithm.

1. A versatile mesh generator is *Gmsh* (www.geuz.org/gmsh/), a program available under the GNU license agreement that is capable of producing unstructured volumetric grids. It is our tool of choice for mesh generation.

2. We often make use of *ParaView* (www.paraview.org) for visualizing vector fields.

III. FVTD for Conservation Laws and Maxwell's Equations

The FVTD algorithm is usually applied to physical phenomena which are governed by a *conservation law*. For example, given a scalar quantity, denoted by $u(\boldsymbol{x}, t)$, a typical conservation law would be,

$$\partial_t u(\boldsymbol{x}, t) + \nabla \cdot \boldsymbol{f}(u(\boldsymbol{x}, t)) = S(\boldsymbol{x}, t) \tag{1}$$

where the flux vector $\boldsymbol{f}$ is some function of u, and $S(\boldsymbol{x}, t)$ is a source term. Integrating the conservation law over an arbitrary volume, T_i, with boundary ∂T_i gives,

$$\int_{T_i} \partial_t u(\boldsymbol{x}, t) dV + \int_{\partial T_i} \boldsymbol{f}(u(\boldsymbol{x}, t)) \cdot d\boldsymbol{s} = \int_{T_i} S(\boldsymbol{x}, t) dV \tag{2}$$

where the divergence theorem has been applied to the second term and $d\boldsymbol{s} = \hat{\boldsymbol{n}} ds$ is the outward directed surface element vector. The FVTD method for solving electromagnetic problems considers all of the electric and magnetic field components as components of a solution vector $\boldsymbol{u} = [\ \boldsymbol{E}\ \boldsymbol{H}\ ]^{\mathrm{T}}$, and then casts Maxwell's equations into a form analogous to (1). Following a procedure similar to that given in [3], starting from Maxwell's two curl equations,

$$\begin{cases} \varepsilon \partial_t \boldsymbol{E} - \nabla \times \boldsymbol{H} + \sigma \boldsymbol{E} = -\boldsymbol{J} \\ \mu \partial_t \boldsymbol{H} + \nabla \times \boldsymbol{E} = 0 \end{cases} \tag{3}$$

we employ the matrix operator,

$$\boldsymbol{S}(\boldsymbol{x})\boldsymbol{b} = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \boldsymbol{x} \times \boldsymbol{b} \tag{4}$$

by which the curl of a vector can be expressed in terms of the divergence of a matrix operating on the vector,

$$\nabla \times \boldsymbol{x} \equiv (\mathrm{div}\boldsymbol{S}(\boldsymbol{x}))^{\mathrm{T}} = \begin{bmatrix} \partial_2 x_3 - \partial_3 x_2 \\ -\partial_1 x_3 + \partial_3 x_1 \\ \partial_1 x_2 - \partial_2 x_1 \end{bmatrix}. \tag{5}$$

In terms of this new operator, Maxwell's equations can be written as

$$\begin{cases} \varepsilon \partial_t \boldsymbol{E} - (\mathrm{div}\boldsymbol{S}(\boldsymbol{H}))^{\mathrm{T}} + \sigma \boldsymbol{E} = -\boldsymbol{J} \\ \mu \partial_t \boldsymbol{H} + (\mathrm{div}\boldsymbol{S}(\boldsymbol{E}))^{\mathrm{T}} = 0 \end{cases} \tag{6}$$

or, even more succinctly as

$$\partial_t \boldsymbol{u} + \alpha^{-1} \boldsymbol{K}\boldsymbol{u} = \alpha^{-1}(\boldsymbol{G} + \boldsymbol{B}\boldsymbol{u}), \tag{7}$$

where,

$$\alpha \triangleq \begin{bmatrix} \varepsilon & \boldsymbol{0} \\ \boldsymbol{0} & \mu \end{bmatrix},\ \boldsymbol{K}\boldsymbol{u} = \begin{bmatrix} -\nabla \times \boldsymbol{H} \\ \nabla \times \boldsymbol{E} \end{bmatrix},\ \boldsymbol{B} \triangleq \begin{bmatrix} -\sigma & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix},\ \boldsymbol{G} \triangleq \begin{bmatrix} -\boldsymbol{J} \\ \boldsymbol{0} \end{bmatrix}. \tag{8}$$

Integrating the curl equations (7) over an element denoted by T_i with boundary ∂T_i and using the

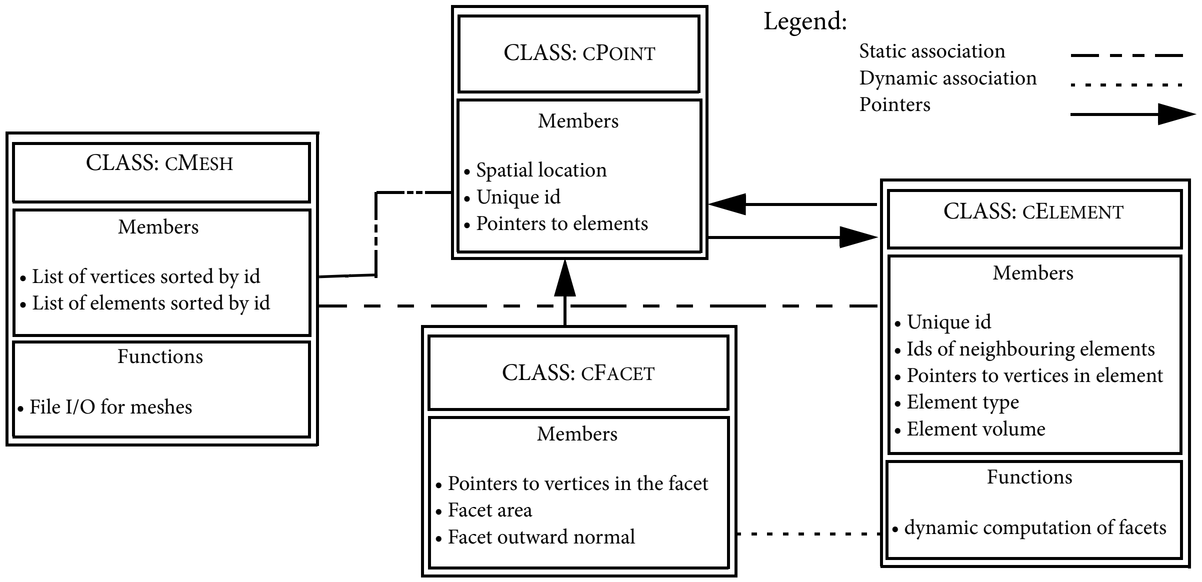


Fig. 1. Object-oriented mesh representation: class hierarchy.

divergence theorem to convert the integral over volume to integrals over the surface as in (2) we arrive at

$$\int_{T_i} \partial_t \boldsymbol{u} d\boldsymbol{x} + \int_{\partial T_i} \alpha^{-1}\boldsymbol{A}(\hat{\boldsymbol{n}})\boldsymbol{u} ds = \int_{T_i} \alpha^{-1}\boldsymbol{G} d\boldsymbol{x} + \int_{T_i} \alpha^{-1}\boldsymbol{B}\boldsymbol{u} d\boldsymbol{x} \quad (9)$$

where matrix $\boldsymbol{A}(\hat{\boldsymbol{n}})$ is defined by,

$$\boldsymbol{A}(\hat{\boldsymbol{n}}) = \begin{bmatrix} \boldsymbol{0} & -\boldsymbol{S}(\hat{\boldsymbol{n}}) \\ \boldsymbol{S}(\hat{\boldsymbol{n}}) & \boldsymbol{0} \end{bmatrix}, \quad (10)$$

where $\hat{\boldsymbol{n}}$ denotes the outward normal to the volume surface ∂T_i. Analogous to (2), the value $\boldsymbol{A}(\hat{\boldsymbol{n}})\boldsymbol{u}$ is referred to as the flux through the surface ∂T_i.

In order to discretize the electromagnetic problem of interest we associate with each cell a value of the generalized solution vector $\boldsymbol{u}_i$ located at the barycentre $\boldsymbol{x}_i$ of element T_i. This value is taken to represent the average of the generalized solution vector over the element T_i, *i.e.*,

$$\boldsymbol{u}_i = \frac{1}{\mu(T_i)} \int_{T_i} \boldsymbol{u}(\boldsymbol{x}) d\boldsymbol{x} = \boldsymbol{u}(\boldsymbol{x}_i) + O(\|\Delta \boldsymbol{x}^2\|) \quad (11)$$

where $\mu(T_i)$ is the volume of element T_i, and where $\Delta\boldsymbol{x}$ scales with the size of the element. Next, we define $\tilde{\boldsymbol{A}}(\hat{\boldsymbol{n}}) \triangleq \alpha^{-1}\boldsymbol{A}(\hat{\boldsymbol{n}})$ and decompose $\tilde{\boldsymbol{A}}(\hat{\boldsymbol{n}})$ as a sum of matrices with positive and negative eigenvalues (due to the symmetry of $\boldsymbol{A}(\hat{\boldsymbol{n}})$ the eigenvalues are real). Limiting each volumetric element to a homogeneous isotropic space described by material parameters $\varepsilon = \mathrm{diag}(\varepsilon, \varepsilon, \varepsilon)$ and $\mu = \mathrm{diag}(\mu, \mu, \mu)$, it can be shown that the matrix $\tilde{\boldsymbol{A}}(\hat{\boldsymbol{n}})$ has six eigenvalues given by $\Lambda = \mathrm{diag}\{0, 0, v, v, -v, -v\}$ where $v = 1/\sqrt{\varepsilon\mu}$. To avoid inducing artificial numerical oscillations into the solution, it is beneficial to perform *flux-splitting* [3] by decomposing $\tilde{\boldsymbol{A}}(\hat{\boldsymbol{n}})$ into

$$\tilde{\boldsymbol{A}}(\hat{\boldsymbol{n}}) = \tilde{\boldsymbol{A}}(\hat{\boldsymbol{n}})^{+} + \tilde{\boldsymbol{A}}(\hat{\boldsymbol{n}})^{-}, \quad (12)$$

where,

$$\tilde{\boldsymbol{A}}(\hat{\boldsymbol{n}})^{+} = \frac{1}{2}\begin{bmatrix} -v\boldsymbol{S}(\hat{\boldsymbol{n}})^2 & -\varepsilon^{-1}\boldsymbol{S}(\hat{\boldsymbol{n}}) \\ \mu^{-1}\boldsymbol{S}(\hat{\boldsymbol{n}}) & -v\boldsymbol{S}(\hat{\boldsymbol{n}})^2 \end{bmatrix}$$

$$\tilde{\boldsymbol{A}}(\hat{\boldsymbol{n}})^{-} = \frac{1}{2}\begin{bmatrix} v\boldsymbol{S}(\hat{\boldsymbol{n}})^2 & -\varepsilon^{-1}\boldsymbol{S}(\hat{\boldsymbol{n}}) \\ \mu^{-1}\boldsymbol{S}(\hat{\boldsymbol{n}}) & v\boldsymbol{S}(\hat{\boldsymbol{n}})^2 \end{bmatrix}. \quad (13)$$

To compute the value of the surface integral in (9) we require knowledge of the flux $\boldsymbol{A}(\hat{\boldsymbol{n}})\boldsymbol{u}$ on the boundary ∂T_i. To determine the flux, we let $\boldsymbol{u}^{*}$ denote the solution vector on the inside part of the surface ∂T_i while using $\boldsymbol{u}^{**}$ to denote the solution vector on the outside part of ∂T_i. We can consider $\boldsymbol{u}^{*}$ and $\boldsymbol{u}^{**}$ as limits of the solution $\boldsymbol{u}(\boldsymbol{x})$ from the inside or outside of the element on to ∂T_i. The electromagnetic boundary conditions for the continuity of the tangential electric and magnetic field components across a boundary,

$$\begin{cases} \hat{\boldsymbol{n}} \times \boldsymbol{E}^{*} = \hat{\boldsymbol{n}} \times \boldsymbol{E}^{**} = \hat{\boldsymbol{n}} \times \boldsymbol{E} \\ \hat{\boldsymbol{n}} \times \boldsymbol{H}^{*} = \hat{\boldsymbol{n}} \times \boldsymbol{H}^{**} = \hat{\boldsymbol{n}} \times \boldsymbol{H} \end{cases}, \quad (14)$$

are used to express the flux at the surface ∂T_i in terms of the operator $\boldsymbol{A}(\hat{\boldsymbol{n}})$ as $\boldsymbol{A}(\hat{\boldsymbol{n}})\boldsymbol{u} = \boldsymbol{A}(\hat{\boldsymbol{n}})\boldsymbol{u}^{*} = \boldsymbol{A}(\hat{\boldsymbol{n}})\boldsymbol{u}^{**}$. As $\boldsymbol{u}^{*}$ and $\boldsymbol{u}^{**}$ are not known explicitly they must be interpolated from the known values in the cell interior.

The flux at the boundary ∂T_i may be split by first calculating $\hat{\boldsymbol{n}} \times \boldsymbol{E}$ and $\hat{\boldsymbol{n}} \times \boldsymbol{H}$ at the cell boundary. Let us consider the top and bottom blocks of $\tilde{\boldsymbol{A}}(\hat{\boldsymbol{n}})^{+}\boldsymbol{u}^{*}$ and $\tilde{\boldsymbol{A}}(\hat{\boldsymbol{n}})^{-}\boldsymbol{u}^{**}$ separately. The top block is

$$-\frac{1}{2}v_1 S(\hat{\boldsymbol{n}})^2\boldsymbol{E}^{*} - \frac{1}{2}\frac{S(\hat{\boldsymbol{n}})}{\varepsilon_1}\boldsymbol{H}^{*} \quad \text{and}$$

$$\frac{1}{2}v_2 S(\hat{\boldsymbol{n}})^2\boldsymbol{E}^{**} - \frac{1}{2}\frac{S(\hat{\boldsymbol{n}})}{\varepsilon_2}\boldsymbol{H}^{**}. \quad (15)$$

We multiply the first by ε_1 and the second by ε_2 to obtain

$$-\frac{1}{2}Y_1 S(\hat{\boldsymbol{n}})^2\boldsymbol{E}^{*} - \frac{1}{2}S(\hat{\boldsymbol{n}})\boldsymbol{H}^{*} \text{ and}$$

$$\frac{1}{2}Y_2 S(\hat{\boldsymbol{n}})^2\boldsymbol{E}^{**} - \frac{1}{2}S(\hat{\boldsymbol{n}})\boldsymbol{H}^{**} \quad (16)$$

where $Y_s = \varepsilon_s v_s = \sqrt{\varepsilon_s/\mu_s}$. Finally, a linear combination of these two gives the desired quantity,

$$\begin{aligned} \hat{\boldsymbol{n}} \times \boldsymbol{H} = {} & \alpha\left(-\frac{1}{2}Y_1 S(\hat{\boldsymbol{n}})^2\boldsymbol{E}^{*} - \frac{1}{2}S(\hat{\boldsymbol{n}})\boldsymbol{H}^{*}\right) + \\ & \beta\left(\frac{1}{2}Y_2 S(\hat{\boldsymbol{n}})^2\boldsymbol{E}^{**} - \frac{1}{2}S(\hat{\boldsymbol{n}})\boldsymbol{H}^{**}\right) \end{aligned} \quad (17)$$

where $\alpha = 2Y_2/(Y_1 + Y_2)$ and $\beta = 2Y_1/(Y_1 + Y_2)$. A similar construction can be made for the bottom block of $\tilde{\boldsymbol{A}}(\hat{\boldsymbol{n}})^{+}\boldsymbol{u}^{*}$ and $\tilde{\boldsymbol{A}}(\hat{\boldsymbol{n}})^{-}\boldsymbol{u}^{**}$ in order to obtain $\hat{\boldsymbol{n}} \times \boldsymbol{E}$.

Using these results, we come to a concise representation for the flux,

$$\boldsymbol{A}(\hat{\boldsymbol{n}})\boldsymbol{u} = \boldsymbol{T}_i\alpha_i\tilde{\boldsymbol{A}}(\hat{\boldsymbol{n}})^{+}\boldsymbol{u}^{*} + \boldsymbol{T}_k\alpha_k\tilde{\boldsymbol{A}}(\hat{\boldsymbol{n}})^{-}\boldsymbol{u}^{**} \quad (18)$$

where,

$$T_i = 2\begin{bmatrix} Y_k(Y_i+Y_k)^{-1}\boldsymbol{I} & 0 \\ 0 & Z_k(Z_i+Z_k)^{-1}\boldsymbol{I} \end{bmatrix} \text{ and}$$

$$T_k = \begin{bmatrix} Y_i(Y_i+Y_k)^{-1}\boldsymbol{I} & 0 \\ 0 & Z_i(Z_i+Z_k)^{-1}\boldsymbol{I} \end{bmatrix} \qquad (19)$$

and $\boldsymbol{I} \in \mathbb{R}^{3\times 3}$ is the identity matrix, and $Z = Y^{-1}$ is the impedance.

At PEC boundaries zero tangential electric field and the image principle can be used to derive that the linear operation of $\tilde{A}(\hat{n})$ on u becomes $\tilde{A}(\hat{n})u = \alpha_i T^{\mathrm{pc}}\tilde{A}(\hat{n})^+ u^*$, where

$$T^{\mathrm{pc}} = \begin{bmatrix} 2\boldsymbol{I} & 0 \\ 0 & 0 \end{bmatrix}. \qquad (20)$$

Consequently, it can be shown that the scattered field formulation derived using the image principle results in,

$$\tilde{A}(\hat{n})u^s = \alpha_i T^{\mathrm{pc}}\tilde{A}(\hat{n})^+ u^{s*} - 2\tilde{A}(\hat{n})^- G^{s*}, \qquad (21)$$

where $G^{s*} = \begin{bmatrix} -E^i & 0 \end{bmatrix}^{*\mathrm{T}}$ is the source term for the scattered field formulation.

Finally, while a discussion of high-order mesh truncation schemes is beyond the scope of this paper, it is possible to obtain a simple mesh boundary condition by setting $\tilde{A}(\hat{n})^-$ to zero at the mesh boundary.

IV. Flux Integration: MUSCL and Polynomial Interpolation

It is apparent from (9) and (18) that integration around the boundary of an element requires knowledge of the flux (or equivalently the solution values u^* and u^{**}) at both sides of the cell boundary. As we are only storing the solution at the barycentres of the elements, we require interpolation of these values to the element border in order to accurately integrate the flux. Two common techniques for interpolating the flux at the cell boundary are the so-called *upwind* and MUSCL schemes [3]. As upwinding provides only first-order accuracy and results in significant dissipation, we omit it from further consideration. For brevity, details of the MUSCL scheme are omitted, however the MUSCL scheme as detailed in [3] was implemented yielding second-order accurate results as will be shown in Section 6. Further, we have applied polynomial interpolation, for which we now summarize the required theory for computing the fluxes at the volumetric element boundaries.

IV.A The ENO Requirement

In any interpolatory technique used to compute the value of the solution u at the boundary of a facet, we make use of a stencil comprised of the values u_i located at the centers of some elements in the neighbourhood of the facet of interest. Although stencils for an arbitrarily high order of approximation are available, when they're applied to a solution with strong gradients, experience in CFD has shown that this could result in unwanted numerical oscillations [7]. The idea behind *essentially non-oscillatory* (ENO) interpolation schemes, used frequently in CFD for approximating the solution value at a given facet, is to use only neighbouring solution values that are *smoothly* connected to the solution at the facet in question. That is, the stencil that we use to approximate the facet values cannot cross points where the solution has steep gradients. Details can be found in several publications dealing with computational fluid dynamics (see, for example, [7, 8]) which deal with problems that involve the evolution of shocks.

Initially, we believed that maintaining ENO interpolation schemes would be critical to updating the solution inside the computational domain. We thought that material boundaries would require special care in selecting the interpolation stencil in an analogous manner to handling shocks in computational fluid dynamics. However, upon numerical experimentation, we found that such interpolatory stencil modifications at material boundaries were unnecessary and, due to the overhead required in computing the modified stencils, we no longer impose ENO requirements. To date, we have not encountered a problem of interest where such a scheme is required. We do however, acknowledge that for some problems, we may have to re-evaluate the importance of ENO schemes.

IV.B General Polynomial Interpolation

The work described herein closely follows the work of Ollivier-Gooch found in [8]. In describing the polynomial interpolation method of Ollivier-Gooch we consider a polynomial $P_i(x - x_i)$ which interpolates the field value solution $u(x)$ around a point x_i and require that the difference between this polynomial and the exact solution be of order $k + 1$, that is,

$$P_i(x - x_i) - u(x) = O(\|x - x_i\|^{k+1}). \qquad (22)$$

IV.C Stencil Selection

We require a stencil such that the data we use to determine the coefficients of our polynomial are not too far from the control volume center, x_i. A good way to collect finite volumes for the stencil around a finite volume T_i is to use the first neighbouring elements around T_i, *i.e.,* those which share a common facet with T_i. Iterating this process and thereby adding new elements to our control volume, we will collect a large enough stencil to achieve the desired accuracy: enough points to determine the coefficients for our polynomial. The set of elements from which the stencil will be generated may be written as,

$$S_i^k = \{T_s : \exists T_n \in S_i^{k-1}, \partial T_s \cap \partial T_n \neq \varnothing\} \tag{23}$$

with $S_i^1 = \{T_i\}$. The intersection $\partial T_s \cap \partial T_n$ is assumed null if there is no common facet between elements T_s and T_n. The stencil S_i^k is then built by considering the center points of each element in S_i^{k-1}. Usually k will be the same as the maximal possible order of accuracy for the stencil [8].

IV.D Polynomial Interpolation Theory

According to the dimensions of our problem, we consider the function $\boldsymbol{u}(\boldsymbol{x}) : \mathbb{R}^3 \to \mathbb{R}^6$ and we write a general Taylor's expansion about the point $\boldsymbol{x}_i$. If the function is infinitely differentiable in the neighbourhood of $\boldsymbol{x}_i$, then we can write,

$$\boldsymbol{u}(\boldsymbol{x}) = \sum_{m=0}^{k} \frac{1}{m!} \sum_{m_1, m_2, m_3} \frac{\partial^m \boldsymbol{u}(\boldsymbol{x}_i)}{(\partial x_1)^{m_1}(\partial x_2)^{m_2}(\partial x_3)^{m_3}} \times \prod_{s=1}^{3} (x_s - (x_i)_s)^{m_s} + O(\|\Delta \boldsymbol{x}\|^{k+1}) \tag{24}$$

which is a polynomial approximation for $\boldsymbol{u}(\boldsymbol{x})$ given by the truncated Taylor's expansion of degree k. In (24) $m_1 + m_2 + m_3 = m$. As the FVTD solution is in terms of cell-averaged values $\boldsymbol{u}_i$, we let the expansion point $\boldsymbol{x}_i$ correspond to the barycentre of T_i and take a volumetric average of both sides of (24) over T_i to obtain,

$$\frac{1}{\mu(T_i)} \int_{T_i} \boldsymbol{u}(\boldsymbol{x}) d\boldsymbol{x} \triangleq \boldsymbol{u}_i = \sum_{m=0}^{k} \frac{1}{m!} \sum_{m_1, m_2, m_3} \frac{\partial^m \boldsymbol{u}(\boldsymbol{x}_i)}{(\partial x_1)^{m_1}(\partial x_2)^{m_2}(\partial x_3)^{m_3}} \times \frac{1}{\mu(T_i)} \int_{T_i} \prod_{s=1}^{3} (x_s - (x_i)_s)^{m_s} d\boldsymbol{x} + \frac{1}{\mu(T_i)} \int_{T_i} O(\|\Delta \boldsymbol{x}\|^{k+1}) d\boldsymbol{x} \tag{25}$$

Now it is our desire to express $\boldsymbol{u}(\boldsymbol{x})$ in terms of the average values $\boldsymbol{u}_i$. We begin by extracting the first term of the summation in (25) which happens to be $\boldsymbol{u}(\boldsymbol{x}_i)$ and write it in terms of $\boldsymbol{u}_i$ as,

$$\boldsymbol{u}(\boldsymbol{x}_i) = \boldsymbol{u}_i - \sum_{m=1}^{k} \frac{1}{m!} \sum_{m_1, m_2, m_3} \frac{\partial^m \boldsymbol{u}(\boldsymbol{x}_i)}{(\partial x_1)^{m_1}(\partial x_2)^{m_2}(\partial x_3)^{m_3}} \times \frac{1}{\mu(T_i)} \int_{T_i} \prod_{s=1}^{3} (x_s - (x_i)_s)^{m_s} d\boldsymbol{x} - \frac{1}{\mu(T_i)} \int_{T_i} O(\|\Delta \boldsymbol{x}\|^{k+1}) d\boldsymbol{x} \tag{26}$$

Next, by extracting the first term of (24) we obtain

$$\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{u}(\boldsymbol{x}_i) + \sum_{m=1}^{k} \frac{1}{m!} \sum_{m_1, m_2, m_3} \frac{\partial^m \boldsymbol{u}(\boldsymbol{x}_i)}{(\partial x_1)^{m_1}(\partial x_2)^{m_2}(\partial x_3)^{m_3}} \times \prod_{s=1}^{3} (x_s - (x_i)_s)^{m_s} + O(\|\Delta \boldsymbol{x}\|^{k+1}). \tag{27}$$

Finally, substitution for $\boldsymbol{u}(\boldsymbol{x}_i)$ from (26) into (27) gives,

$$\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{u}_i + \sum_{m=1}^{k} \frac{1}{m!} \sum_{m_1, m_2, m_3} \frac{\partial^m \boldsymbol{u}(\boldsymbol{x}_i)}{(\partial x_1)^{m_1}(\partial x_2)^{m_2}(\partial x_3)^{m_3}} \times \left[\prod_{s=1}^{3} (x_s - (x_0)_s)^{m_s} - p_i(m_1, m_2, m_3) \right] + O(\|\Delta \boldsymbol{x}\|^{k+1}) - \frac{1}{\mu(T_i)} \int_{T_i} O(\|\Delta \boldsymbol{x}\|^{k+1}) d\boldsymbol{x} \tag{28}$$

where

$$p_i(m_1, m_2, m_3) \triangleq \frac{1}{\mu(T_i)} \int_{T_i} \prod_{s=1}^{3} (x_s - (x_0)_s)^{m_s} d\boldsymbol{x} \tag{29}$$

define the element moments and where $\boldsymbol{P}_i(\boldsymbol{x} - \boldsymbol{x}_i)$ corresponds to the first two terms on the right hand side of (28) because the two error terms are of the same order. By the selected substitution method we have implicitly enforced that the average polynomial value over the element T_i is equal to the cell-averaged solution $\boldsymbol{u}_i$ *i.e.*,

$$\frac{1}{\mu(T_i)} \int_{T_i} \boldsymbol{P}_i(\boldsymbol{x} - \boldsymbol{x}_i) d\boldsymbol{x} = \frac{1}{\mu(T_i)} \int_{T_i} \boldsymbol{u}(\boldsymbol{x}) d\boldsymbol{x} \triangleq \boldsymbol{u}_i, \tag{30}$$

due to the fact that, upon cell-averaging of (28), the error terms cancel. Finally, to determine the polynomial coefficients, corresponding to the partial derivatives in (28), we minimize,

$$\left\| \boldsymbol{u}_j - \frac{1}{\mu(T_j)} \int_{T_j} \boldsymbol{P}_i(\boldsymbol{x}-\boldsymbol{x}_i) d\boldsymbol{x} \right\| \qquad \forall T_j \in S_i^k, \tag{31}$$

where details can be found in [8].

We note that expression (28) can be used to compute the true solution anywhere within the stencil from the cell-averaged value in each element. Therefore, this method enables us to convert the stored cell-average values into a high-order solution for $\boldsymbol{u}^*$ and $\boldsymbol{u}^{**}$ on element boundaries. It also allows us to output the true solution for electromagnetic problems of interest such as for scattering from a PEC sphere as shown in Section 6.

IV.E Object-Oriented Flux Interpolation

It is clear that an object-oriented approach to the implementation of an engine capable of various flux-interpolation schemes is possible. Further, one may be interested in adaptively applying different flux-integration techniques to save on computational resources in regions where high-order techniques are not required for adequate accuracy. Therefore, it is not desirable to derive a flux-interpolation method from a base class. Instead, our implementation uses a switch statement to select between the methods so that all methods are at our disposal during computation.

Using the above methods to approximate the flux at the boundaries *i.e.* $\boldsymbol{u}^*$ and $\boldsymbol{u}^{**}$ we substitute (18) into the second term of (9),

$$\frac{1}{\mu(T_i)} \int_{\partial T_i} \alpha^{-1} \boldsymbol{A}(\hat{\boldsymbol{n}}) \boldsymbol{u} ds \approx \\ \frac{1}{\mu(T_i)} \sum_{m=1}^{M_i} (\boldsymbol{T}_i \alpha_i \tilde{\boldsymbol{A}}(\hat{\boldsymbol{n}}_m)^+ \tilde{\boldsymbol{u}}^* + \boldsymbol{T}_{i_m} \alpha_{i_m} \tilde{\boldsymbol{A}}(\hat{\boldsymbol{n}}_m)^- \tilde{\boldsymbol{u}}^{**}) \tag{32}$$

where M_i is the number of facets making up element T_i, i_m denotes the element neighbouring T_i via its mth facet and $\hat{\boldsymbol{n}}_m$ denotes the outward normal to the mth facet. Note that $\tilde{\boldsymbol{u}}^*$ and $\tilde{\boldsymbol{u}}^{**}$ are the integrals of the inner and outer solutions over facet m respectively which we compute analytically from the polynomial interpolation function up to third-order accuracy.

Finally, for a mesh comprising of N elements we define

$$\begin{bmatrix} \left(\frac{1}{V_1} \int_{\partial T_1} \alpha^{-1} \boldsymbol{A}(\hat{\boldsymbol{n}}) \boldsymbol{u} ds \right)^T \\ \vdots \\ \left(\frac{1}{V_N} \int_{\partial T_N} \alpha^{-1} \boldsymbol{A}(\hat{\boldsymbol{n}}) \boldsymbol{u} ds \right)^T \end{bmatrix} \triangleq \boldsymbol{L}\boldsymbol{U} \tag{33}$$

where $\boldsymbol{U} = \begin{bmatrix} \boldsymbol{u}_1^T & \boldsymbol{u}_2^T & \dots & \boldsymbol{u}_N^T \end{bmatrix}^T$ is the vector of all unknowns. Using the notation of (32) and (33) in (9) gives,

$$\partial_t \boldsymbol{U} + \boldsymbol{L}\boldsymbol{U} = \boldsymbol{F} \tag{34}$$

where the time-derivative is taken element by element over $\boldsymbol{U}$ and where $\boldsymbol{F}$ is a source term where each element of $\boldsymbol{F}$ represents the right-hand-side of (9) at the ith cell. It is of importance to note that under linear flux interpolation the result of the operator $\boldsymbol{L}$ operating on $\boldsymbol{U}$ can be viewed as a matrix-vector product.

V. Time-Integration Schemes

Having organized the flux-integration into a matrix-vector product over the entire computational space, it remains to discretely approximate the time derivative in (34). We have considered forward-Euler, predictor-corrector, Runge-Kutta and Crank-Nicholson methods. These methods are explicit schemes, save Crank-Nicholson. For source-free media, the predictor-corrector scheme discretization of (34) using a time-step of Δt will give a system of equations of the form,

$$\boldsymbol{U}^{(n+1)} = \boldsymbol{U}^{(n)} - \Delta t \boldsymbol{L} \boldsymbol{U}^{(n)} + \frac{\Delta t^2}{2} \boldsymbol{L}^2 \boldsymbol{U}^{(n)} + O(|\Delta t|^2). \tag{35}$$

It is clear that this may be re-written as,

$$\boldsymbol{U}^{(n+1)} = \tilde{\boldsymbol{L}} \boldsymbol{U}^{(n)} \tag{36}$$

where, $\tilde{\boldsymbol{L}} = \boldsymbol{I} - \Delta t \boldsymbol{L} + (\Delta t^2/2)\boldsymbol{L}^2$ and $\boldsymbol{I} \in \mathbb{R}^{6N \times 6N}$. In fact, any explicit scheme can be formulated as a matrix-vector-product and so an explicit formulation of the FVTD method requires the one-time filling of the matrix $\tilde{\boldsymbol{L}}$, setting the initial value of the solution vector over the domain, followed by a simple matrix-vector product at each time-step. Implementation of this update scheme should make use of the matrix-vector product updating inherent in the algorithm. For this reason, we have used our matrix library, based on an underlying abstract matrix class, namely, *cAbstractMatrix.* The benefit is that this abstract class is compatible with various linear system solvers such as our own implementations of *GMRES*, *BCGStab,* etc. [9], so that in implicit schemes, when the operator shows up on the left-hand side of the update equation, our solvers can be used.

Unfortunately, storing the entire operator matrix will require substantial amounts of memory for large meshes. For example, in the case of a tetrahedral mesh using simple upwinding, each block of six rows of the matrix would contain five (corresponding to the element in question and its four facets) 6×6 blocks. A mesh consisting of one million elements where 8 byte double precision values are used to store the matrix entries would require $6 \times 6 \times 5 \times 8 \times 10^6 = 1.44$ Gigabytes of RAM to store the matrix. The situation becomes worse when higher-order interpolatory schemes are considered (for example the MUSCL scheme would increase this requirement 5 times). With our current resources we are unable to store the entire matrix. Instead, we update the solution $\boldsymbol{u}_j$ by dynamically computing the i^{th} six-row block of $\boldsymbol{L}$. Therefore, for explicit schemes, we have overloaded the *cAbstractMatrix* matrix-vector-product operator with this row-by-row method of multiplication. The matrix class we use to implement the *cAbstractMatrix* class for solving FVTD problems we call *cMaxwellApprox.* A depiction of the conceptual flow of the algorithm as well as the corresponding object-oriented design is shown in Fig. 2.

VI. Numerical Results and Discussion

VI.A Scattering from a PEC sphere

We present the FVTD results for scattering from a perfectly electrical conducting (PEC) sphere as an exact series solution is available in the frequency domain [10], and a time domain solution may be easily obtained using the inverse Fourier transform. This problem was selected as a benchmark for the FVTD engine as the irregular surface of the sphere coincides with one of the primary reasons for developing finite-volume methods on irregular grids: eliminating the need for stair-stepping at arbitrarily shaped boundaries.

The finite-volume algorithm using the previously discussed flux- and time-integration methods was tested for a PEC sphere with a three metre radius centered at the origin of a Cartesian coordinate system. An x-polarized electric-field plane-wave transient pulse $\boldsymbol{E} = g(t)\hat{\boldsymbol{x}}$ incident in the z-direction and varying as the derivative of a Gaussian was selected where, for $t \geq 0$

$$g(t) = -2A(t-t_0)b^{-2}\exp(-b^{-2}(t-t_0)^2). \qquad (37)$$

The parameters were selected as: $A = 1$, $b = 1.14 \times 10^{-8}$ s, and $t_0 = 4.0 \times 10^{-8}$ s giving a shortest free-space wavelength of about 3 metres resulting in significant energy in the resonance region of the sphere.

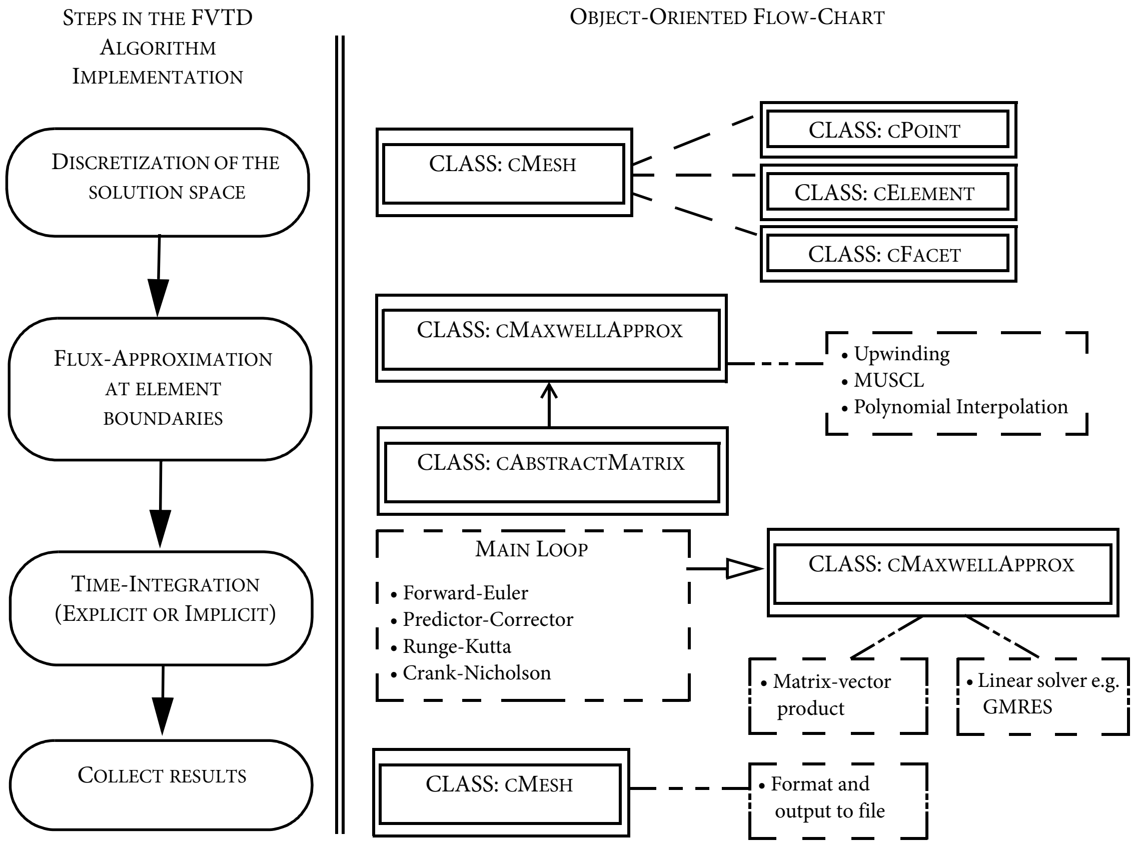

Steps in the FVTD Algorithm Implementation
Object-Oriented Flow-Chart
Discretization of the solution space
Flux-Approximation at element boundaries
Time-Integration (Explicit or Implicit)
Collect results
CLASS: cMesh
CLASS: cPoint
CLASS: cElement
CLASS: cFacet
CLASS: cMaxwellApprox
• Upwinding
• MUSCL
• Polynomial Interpolation
CLASS: cAbstractMatrix
Main Loop
• Forward-Euler
• Predictor-Corrector
• Runge-Kutta
• Crank-Nicholson
CLASS: cMaxwellApprox
• Matrix-vector product
• Linear solver e.g. GMRES
CLASS: cMesh
• Format and output to file

Fig. 2. Algorithm flow and associated class description for the FVTD engine.

Fig. 3 shows the computational results for the x-component of the scattered field at the back-scatter location (0, 0, -7) (a), (c), (e), as well as at the side-scatter location (-7, 0, 0) (b), (d), (f). For both measurement locations, the analytic solution is compared to solutions obtained using the MUSCL and polynomial interpolation finite-volume methods computed on a mesh with an average cell edge length of 0.75 m. All results shown use the second-order predictor-corrector time-integration scheme. In addition, FDTD results are presented for a cell edge length of 0.1 m. The figure shows that the FVTD results are in excellent agreement with the analytic solution.

To further test the FVTD engine, the same PEC sphere problem was solved for volumetric meshes with average element edge lengths of 0.6 and 0.5 m. The back-scattering results are summarized in Table 1. In the table, the L_2 error denotes the percent error between the computed solution and the analytic solution as measured by the L_2 norm. Clearly, for the same level of discretization polynomial interpolation is more accurate than MUSCL as expected due to the higher-order spatial approximations used in the polynomial interpolation scheme.

Comparing the FVTD results with the FDTD results we see that for the 0.1 m FDTD grid, the FVTD algorithm provides better accuracy using MUSCL on the 0.5 m mesh. Third-order polynomial interpolation provided more accurate results than FDTD even in the case of the 0.75 m mesh. The increased accuracy obtained for both the MUSCL and polynomial interpolation schemes comes at the cost of increased memory requirements needed to properly store the unstructured mesh as discussed in Section 2. For example, we require 203 Megabytes of RAM to store 700,000 mesh element solutions using the MUSCL scheme, while FDTD uses roughly the same amount of memory for 8 million grid point solutions. Fortunately, the MUSCL solution at this level of discretization already out-performs the accuracy of FDTD for the simple case of the PEC sphere. For more complicated geometries, we feel that the benefits of any of the FVTD techniques will be even more prominent.

VI.B Scattering From a Dielectric cube

To test the FVTD implementation's ability to handle dielectric bodies and finite conductivities we show the scattering from a lossy, dielectric cube. As there is no known analytic solution for this problem we simply compare the results with the FDTD solution. The cube dimensions were selected as 6 m and the physical parameters were selected as: $\varepsilon_r = 4$, $\mu_r = 1$, $\sigma = 0.01$ S/m. An x-polarized, z-incident, electric-field plane-wave Gaussian transient pulse $\boldsymbol{E} = f(t)\hat{\boldsymbol{x}}$ was selected such that for $t \geq 0$

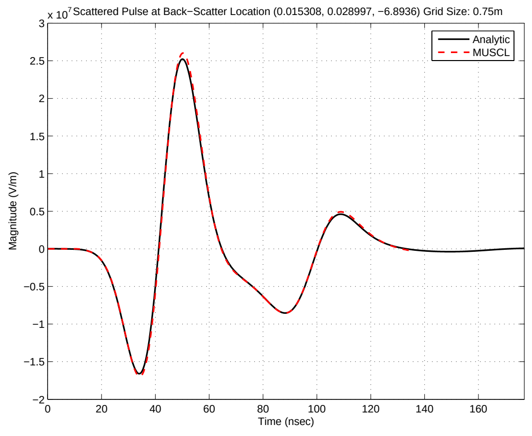
$$f(t) = A\exp(-b^{-2}(t-t_0)^2) \tag{38}$$

with parameters $A = 1000$, $b = 1.14 \times 10^{-8}$ s, and $t_0 = 4.0 \times 10^{-8}$ s. Again, FVTD results were simulated on a volumetric mesh with an average element edge size of 0.75 m and were compared to FDTD results obtained from two regular grids: a 0.2 m regular grid and a 0.5 m regular grid. Results are shown in Fig. 4.

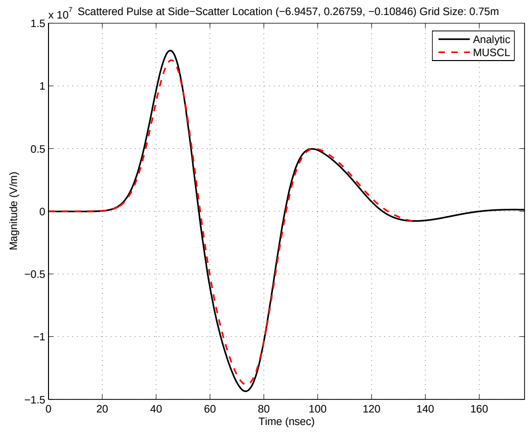
From the results, we can see that the FVTD solution to the lossy cube problem compares quite well with the FDTD solution. The largest difference in the plots is most likely due to differences in the dielectric modelling: FDTD assumes a diamond shaped stencil for imposed dielectric objects while FVTD is capable of imposing a constant dielectric within each finite volume, providing a more accurate physical model of the cube.

Table 1: Comparison of PEC sphere back-scattering results.

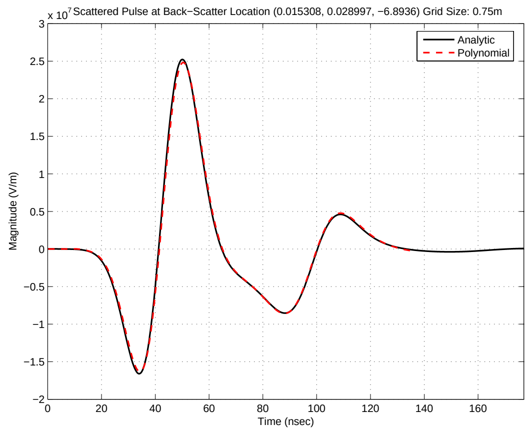
Solution method	Average Element dimension (m)	Number of mesh cells	Memory Require-ments (Mb)	Number of time steps	L_2 Error of E_x
MUSCL	0.75	~200,000	62	~750	7.13%
MUSCL	0.6	~400,000	117	~900	7.40%
MUSCL	0.5	~700,000	203	~1100	2.87%
Polynomial-3	0.75	~200,000	131	~750	3.12%
Polynomial-3	0.6	~400,000	251	~900	1.70%
Polynomial-3	0.5	~700,000	435	~1100	1.69%
FDTD	0.1	~8,000,000	223	~2000	3.25%



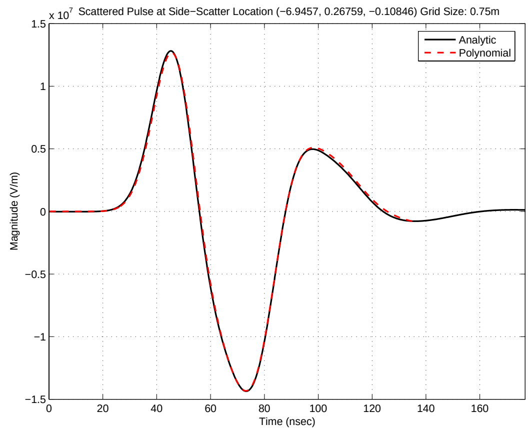
(a)



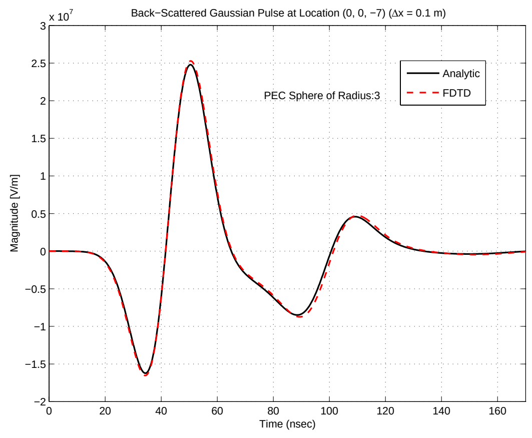
(b)



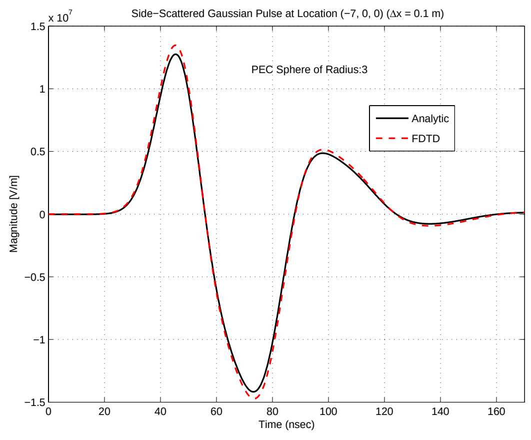
(c)



(d)



(e)



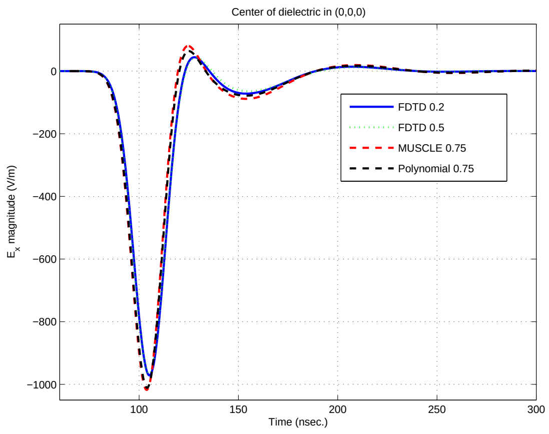
(f)

Fig. 3. Scattering from a 3 meter PEC sphere: Back-scattering (a), (c), (e); and Side-scattering (b), (d), (f); MUSCL (a), (b); Polynomial Interpolation (c), (d); FDTD (e), (f).

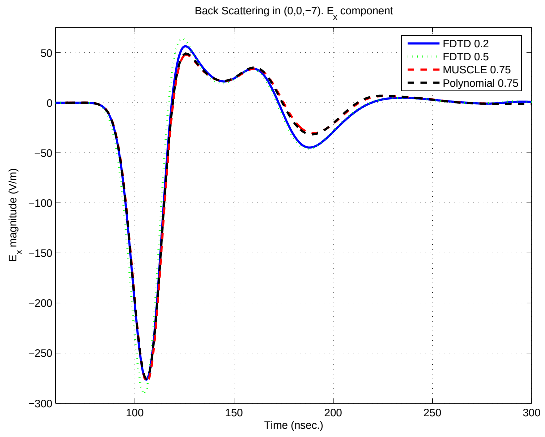
VII. Conclusions

The unstructured grid FVTD method using MUSCL or polynomial interpolated flux-integration, and second-order predictor-corrector time-integration, has given excellent results for all scattering problems tested thus far. The FVTD method has shown to yield more accurate results than FDTD solutions for the same level of discretization. This is due to a more accurate approximation of the geometry for curved scatterers in FVTD. In the case of the PEC sphere, to obtain the same level of accuracy using FDTD required a cubical cell-size 7 times smaller than the average cell edge-length in FVTD.

Although we have not included a comparison of the computational time required by FVTD and FDTD for the same numerical problem, we can say that compared to an un-optimized FDTD implementation, our un-optimized FVTD engine ran substantially faster for the same level of accuracy. We did compare the performance of our engine against a fully optimized commercial FDTD package and found that our engine ran significantly slower. Consequently, we are currently working on optimizing our code and formulating a new criterion for obtaining the maximum time-step which satisfies stability conditions on an unstructured grid.



(a)



(b)

Fig. 4. FVTD and FDTD scattering from a dielectric cube with nonzero conductivity at: cube center (0, 0, 0) (a); back-scatter point (0, 0, -7) (b).

Acknowledgements

The authors would like to acknowledge Defence R&D Canada and NSERC for supporting this research.

References

[1] A. Taflove (editor), *Advances in Computational Electrodynamics: The Finite-Difference Time-Domain Method*, Artech House Inc, Boston, 1998.

[2] V. Shankar, A. H. Mohammadian, and W. F. Hall, "A Time-Domain Finite-Volume Treatment for the Maxwell Equations," *Electromagnetics*, Vol. 10, No. 1-2, pp 127 - 145, 1990.

[3] P. Bonnet, X. Ferrieres, B. L. Michielsen, P. Klotz, and J. L. Roumiguières, "Finite-Volume Time Domain Method," in *Time Domain Electromagnetics*, S. M. Rao (editor), Academic Press, San Diego, 1999.

[4] C. Fumeaux, D. Baumann, P. Leuchtmann, and R. Vahldieck, "A Generalized Local Time-Step Scheme for Efficient FVTD Simulations in Strongly Inhomogeneous Meshes," *IEEE Trans. on Microwave Theory and Techniques*, Vol. 52, No. 3, pp. 1067-1076, March 2004.

[5] C. Hirsch, *Numerical Computation of Internal and External Flows, Vol. I: Fundamentals of Numerical Discretization*, John Wiley & Sons Ltd., New York, 1988.

[6] J. S. Shang, "Characteristic-Based Algorithms for Solving the Maxwell's Equations in the Time Domain," IEEE Antennas and Propagation Magazine, Vol. 37, No. 3, pp. 15-25, June 1995.

[7] X.-D. Liu, S. Osher, and T. Chan, "Weighted Essentially Non-oscillatory Schemes," *Journal of Computational Physics*, vol. 115, pp. 200-212, 1994.

[8] C. F. Ollivier-Gooch, "Quasi-ENO Schemes for Unstructured Meshes Based on Unlimited Data-Dependent Least-Squares Reconstruction," *Journal of Computational Physics*, Vol. 133 (1), pp 6-17, 1997.

[9] Y. Saad, *Iterative Methods for Sparse Linear Systems,* 2nd. Ed. SIAM, 2003.

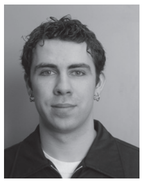
[10] R. F. Harrington, *Time-Harmonic Electromagnetic Fields*, McGraw-Hill, 1961.

[11] D. K. Firsov, I. Jeffrey, J. LoVetri, and C. Gilmore, "An Object Oriented Finite-Volume Time-Domain Computational Engine", *ACES 2006*, *Miami Florida.*

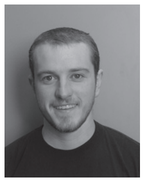
[12] D. Firsov, J. LoVetri, I. Jeffrey, V. Okhmatovski, and W. Chamma, "High-Order FVTD on Unstructured Grids", *ACES 2006, Miami Florida.*

Dmitry K. Firsov received his diploma (with distinction) and Candidate of Science (Ph.D.) degrees in mathematics from Tomsk State University, Russia, in 1999 and 2002 respectively. In 2002 he was an Assistant Professor at Tomsk State University, Department of Theoretical Mechanics. From 2003-2005 he was a post-doctoral fellow with the Department of Mathematics at the University of Manitoba, Canada. Currently, Dmitry is a post-doctoral fellow with the Department of Electrical Engineering at the University of Manitoba. He is currently working in the area of numerical methods for partial differential equations.

Joe LoVetri received the Ph.D. degree in electrical engineering from the University of Ottawa, ON, Canada, in 1991. From 1991 to 1999, he was an Associate Professor in the Department of Electrical and Computer Engineering, University of Western Ontario, Canada. He is currently a Professor in the Department of Electrical and Computer Engineering, and Associate Dean (Research) of the Faculty of Engineering at University of Manitoba, Winnipeg, Canada. His main research interests are in time-domain CEM, modeling of EMC problems, GPR, and inverse imaging techniques.


Ian Jeffrey received the B.Sc degree in Computer Engineering (with distinction) in 2002 and the M.Sc degree in Electrical and Computer Engineering in 2004 from the University of Manitoba, Winnipeg, Canada where he is currently working towards the Ph.D Degree in Electrical and Computer Engineering. His current research interests are in fast algorithms for computational electromagnetics, electromagnetic compatibility and inverse problems.

Vladimir I. Okhmatovski received the M.S. (with distinction) and Candidate of Science (Ph.D.) degrees from the Moscow Power Engineering Institute, Russia, in 1996 and 1997, respectively. From 1998-1999, he was a Post-Doctoral Research Associate with the National Technical University of Athens. From 1999-2003, he was a Post-Doctoral Research Associate with the University of Illinois at Urbana-Champaign. From 2003-2004, he was with the Department of Custom Integrated Circuits Advanced Research and Development, Cadence Design Systems Inc. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB, Canada. His research interests are the fast algorithms of computational electromagnetics and modeling of high-speed digital circuits and mixed-signal components.


Colin Gilmore received the B.Sc degree in Electrical Engineering (with distinction and winning the gold medal) in 2002 and the M.Sc Degree in Electrical Engineering in 2005 both from the University of Manitoba, Winnipeg, Canada where he is currently working towards the Ph.D Degree in Electrical Engineering. His current research interests are in the areas of GPR, signal processing, computational electromagnetics and inverse problems.

Walid Chamma was born in Saida, Lebanon, received the B.E. degree in electrical engineering from the American University of Beirut (AUB), Lebanon, in 1984, and the M. Sc. and Ph.D. degrees in electrical engineering from the University of Manitoba, Winnipeg, MB, Canada, in 1988 and 1993, respectively. From 1993-1998 he worked as a Research Engineer in the Antennas and Electromagnetic Group, InfoMagnetics Technologies (IMT) Corporation, Winnipeg, MB, Canada, where he participated in the design of several low-profile antennas for mobile satellite (MSAT) communications, as well as the development of a general-purpose finite-difference time-domain (FDTD) code for solving electromagnetic problems. In 1999, he joined the Radar Electronic Warfare (REW) section at Defence R&D Canada – Ottawa, as a Defence Scientist. His primary research interest is in applied electromagnetics, antenna design and new concepts in radar applications.