An Object Oriented Finite-Volume Time-Domain Computational Engine

Dmitry K. Firsov, Ian Jeffrey, Joe LoVetri and Colin Gilmore

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

Abstract: An object-oriented implementation of an FVTD engine for solving Maxwell's equations is presented. The relevant aspects of the FVTD method are from an object oriented perspective and details of the object classes are given. An important feature of our FVTD engine is its capability of performing various flux-interpolation schemes which is also discussed. The engine is validated against the series solution for scattering from a PEC sphere.

Keywords: Finite-volume time-domain, Maxwell's equations, object-oriented design.

1. Introduction

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 finite-difference time-domain (FDTD) algorithm for solving Maxwell's equations [1, 2, 3]. 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 structured mesh's ability to accurately describe the physical problem.

In this paper we present the implementation of an object-oriented computational engine for solving Maxwell's Equations on unstructured grids using the FVTD method. The purpose of this paper is two-fold: First we demonstrate the flexibility of an object-oriented implementation of the FVTD algorithm for handling a multitude of volumetric mesh descriptions, time-integration approximations and flux-integration approximations. Second, in order to facilitate an in depth understanding of the method, we provide an outline of the class hierarchy and programming tactics for implementation. Although our FVTD implementation is in C++, the object-oriented concepts we discuss are not language specific.

While the FVTD algorithm has its foundations in computational fluid dynamics, the algorithm may be used to solve any system where the governing physical law is a so-called *conservation law*. This means that at a given spatial location, a given quantity is conserved, *i.e.*, its rate of change is equal to the total flux entering that location. For 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, $S(x, t)$ is a source term, and the divergence of the flux indicates the amount of *outflow* of the quantity at a given point. 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 $\boldsymbol{ds}$ 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 vector $\boldsymbol{u}$ and then casts Maxwell's equations into a form analogous to (1).

The FVTD algorithm proceeds by discretizing the solution domain into non-overlapping volumetric elements and applying (2), or its vector version, on each element. This results in an equation relating $\boldsymbol{u}$ in each element to the flux at the element surface. This relationship can be applied to discretized approximations of $\boldsymbol{u}$. It is clear that two further approximations are required: temporal approximations for the time-derivative and spatial approximations for the flux.

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, while Section 5 overviews the time-integration schemes considered which retain the operator representation of the FVTD update scheme as a matrix-vector-product. Finally, Section 6 validates the engine by comparing computational results with a known solution for the scattering from a PEC sphere.

2. Object Oriented Grid 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 class-instances 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 given a unique ID and contains a list of pointers to *cPoint* objects denoting the vertices of the element as well as a list of the neighbouring element IDs. Storing neighbouring element IDs is essential for an efficient implementation. The element type is specified by a member in the *cElement* class, *e.g.*, a value of 4 is used to indicate a tetrahedron while a value of 5 is used to indicate a hexahedron. 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. Briefly, we note that 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 an object-oriented inheritance to overload abstract functions in the element

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.

class based on the element type. 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.

3. FVTD Formulation of Maxwell's Equations

The FVTD formulation of Maxwell's equations presented here closely follows that of [2]. Using the obvious notation (in SI units), Maxwell's two curl equations are:

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

Employing the 3×3 matrix operator

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

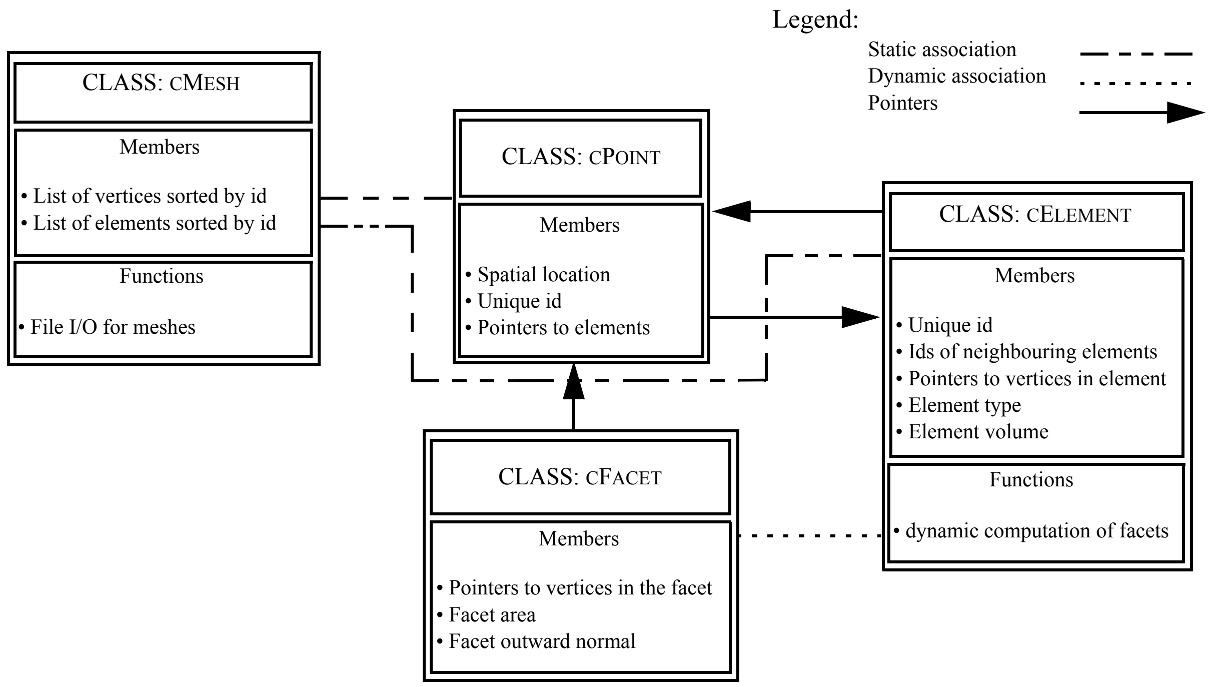

Figure 1. Object-oriented mesh representation: class hierarchy.

the curl of a vector $\boldsymbol{x}$ can be expressed in terms of the divergence of a matrix operating on the vector as follows:

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

where $\mathrm{div}\boldsymbol{S}(\boldsymbol{x})$ is defined to be a row vector whose j^{th} entry is the divergence of the j^{th} column of $\boldsymbol{S}(\boldsymbol{x})$. In terms of this new operator, Maxwell's equations can be written suscinctly as:

$$\partial_t \boldsymbol{u} + \alpha^{-1}\boldsymbol{K}\boldsymbol{u} = \alpha^{-1}(\boldsymbol{G} + \boldsymbol{B}\boldsymbol{u}) \tag{6}$$

where

$$\boldsymbol{u} \triangleq \begin{bmatrix} \boldsymbol{E} \\ \boldsymbol{H} \end{bmatrix},\ \alpha \triangleq \begin{bmatrix} \varepsilon & \boldsymbol{0} \\ \boldsymbol{0} & \mu \end{bmatrix},\ \boldsymbol{K}\boldsymbol{u} = \begin{bmatrix} -(\mathrm{div}\boldsymbol{S}(\boldsymbol{H}))^{\mathrm{T}} \\ (\mathrm{div}\boldsymbol{S}(\boldsymbol{E}))^{\mathrm{T}} \end{bmatrix},\ \boldsymbol{B} \triangleq \begin{bmatrix} -\sigma & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix},\ \boldsymbol{G} \triangleq \begin{bmatrix} -\boldsymbol{J} \\ \boldsymbol{0} \end{bmatrix} \tag{7}$$

and we refer to $\boldsymbol{u}$ as the generalized solution vector. Integrating the curl equations (6) over a volume denoted by T_i with boundary ∂T_i and using the divergence theorem to convert the integral over the volume to integrals over the surface we arrive at

$$\int_{T_i} \partial_t \boldsymbol{u} d\boldsymbol{x} + \int_{\partial T_i} \alpha^{-1}\boldsymbol{A}\boldsymbol{u} ds = \int_{T_i} \alpha^{-1}\boldsymbol{G} d\boldsymbol{x} + \int_{T_i} \alpha^{-1}\boldsymbol{B}\boldsymbol{u} d\boldsymbol{x}. \tag{8}$$

where

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{0} & -\boldsymbol{S}(\hat{\boldsymbol{n}}) \\ \boldsymbol{S}(\hat{\boldsymbol{n}}) & \boldsymbol{0} \end{bmatrix}, \tag{9}$$

and where $\hat{\boldsymbol{n}}$ is the outward directed surface normal.

For the volumetric elements we are considering, ∂T_i is simply a set of N_i plane-facets *i.e.,* $\partial T_i = \bigcup_{n=1}^{N_i} \partial T_{i_n}$ where ∂T_{i_n} denotes the n^{th} facet of element i. Then (8) becomes:

$$\int_{T_i} \partial_t \boldsymbol{u} d\boldsymbol{x} + \sum_{n=1}^{N_i} \mu(\partial T_{i_n})\alpha^{-1}\boldsymbol{A}_n \boldsymbol{u} = \int_{T_i} \alpha^{-1}\boldsymbol{G} d\boldsymbol{x} + \int_{T_i} \alpha^{-1}\boldsymbol{B}\boldsymbol{u} d\boldsymbol{x} \tag{10}$$

where $\mu(\partial T_{i_n})$ is the facet area and $\boldsymbol{A}_n$ is appropriately constructed for the n^{th} facet. Solving the FVTD problem now requires an appropriate time-integration scheme for approximating the time derivative in (10) and knowledge of $\boldsymbol{u}$ at the element surface under appropriate boundary conditions[3]. In order to discretize equation (10) we associate with the volume T_i a value of the generalized solution vector $\boldsymbol{u}_i$ located at the barycentre $\boldsymbol{x}_i$ of the volume. This value is taken to represent the average of the generalized solution vector over the element T_i which corresponds to a second-order approximation to the barycentric value (independent of element geometry). It remains to select appropriate time- and flux-integration schemes.

3. Although it is not difficult to re-formulate (10) for various volumetric boundary conditions, we omit this formulation herein for brevity. For dielectric or PEC interfaces, in addition to a scattered-field formulation, the reader is referred to [2].

4. Flux-Integration Schemes

It is apparent from (10) that integration around the boundary of an element requires knowledge of the flux at the cell boundary. Typically this is accomplished by interpolating from the second-order accurate barycentric values to the facet in question. Most flux-interpolation schemes are based on Taylor's expansion of the generalized solution vector $\boldsymbol{u}$ at the facet in terms of the solution and its derivatives at the element barycentres [4]. The interpolatory schemes we have implemented are upwinding, MUSCL and polynomial interpolation (up to a fifth order of accuracy). To our knowledge, the latter as applied to Maxwell's equations is original work and is discussed in [5]. For details of these schemes the reader is referred to [4].

It is clear that an object-oriented approach to the implementation of an engine capable of various flux-interpolation schemes is possible. Unfortunately, in some instances, (for example at material boundaries) certain high-order schemes result in numerical oscillations [5] and an adaptive interpolatory scheme must be employed. Therefore, it is not desirable to introduce the added complication of overloading an abstract flux-interpolation method in a given class. Instead, our implementation uses a switch statement to select between the methods so that all methods are at our disposal during computation.

For a mesh comprising of N elements we may write the flux-integration at each element in vector form:

$$\left[\left(\frac{1}{V_1}\int_{\partial T_1}\alpha^{-1}\boldsymbol{K}\boldsymbol{u}ds\right)^T\left(\frac{1}{V_2}\int_{\partial T_2}\alpha^{-1}\boldsymbol{K}\boldsymbol{u}ds\right)^T\dots\left(\frac{1}{V_N}\int_{\partial T_N}\alpha^{-1}\boldsymbol{K}\boldsymbol{u}ds\right)^T\right]^T \approx L\boldsymbol{U} \tag{11}$$

where $\boldsymbol{U} = \left[\boldsymbol{u}_1^T\ \boldsymbol{u}_2^T\ \dots\ \boldsymbol{u}_N^T\right]^T$ is the vector of unknowns in each element and where the division by volume V_i arises due to the barycentric averaging of $\boldsymbol{u}_i$ from the left-hand side of (10). Using the result of (11) in (10) gives:

$$\partial_t\boldsymbol{U} + L\boldsymbol{U} = \boldsymbol{F} \tag{12}$$

where the time-derivative is taken element-by-element over $\boldsymbol{U}$ and where $\boldsymbol{F}$ is a source term where each element i of $\boldsymbol{F}$ represents the right-hand-side of (8) at the i^{th} cell. It is of importance to note that under any interpolatory scheme the result of the operator L operating on $\boldsymbol{U}$ can be formulated as a matrix-vector-product.

5. 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 (12). We have considered forward-Euler, predictor-corrector, Runge-Kutta and Crank-Nicolson methods. These methods are explicit schemes, save Crank-Nicolson. For the predictor-corrector scheme, discretization of (12) using a time-step of Δt will give a system of equations of the form:

$$\boldsymbol{U}^{(n+1)} = \boldsymbol{U}^{(n)} - \Delta t L\boldsymbol{U}^{(n)} + \frac{\Delta t^2}{2}L^2\boldsymbol{U}^{(n)} + O(|\Delta t|^2) \tag{13}$$

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

$$\boldsymbol{U}^{(n+1)} = \tilde{L}\boldsymbol{U}^{(n)} \tag{14}$$

where $\tilde{L} = \boldsymbol{I} - \Delta t L + (\Delta t^2/2)L^2$. 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 $\tilde{L}$, 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. [6] so that in implicit schemes, when the operator $\tilde{L}$ shows up on the left-hand side of the update equation, our solvers can be used.

Unfortunately, storing the full 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. For a mesh consisting of one million elements where 8 byte double precision values are used to store the matrix entries would require $6 \times 6 \times 5 \times 8 \times 10^6 = 1.44\ Gb$ of RAM to store the matrix. This is aggravated 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 $\boldsymbol{u}_i$ by dynamically compute the i^{th} six-row block of $\tilde{L}$. 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.

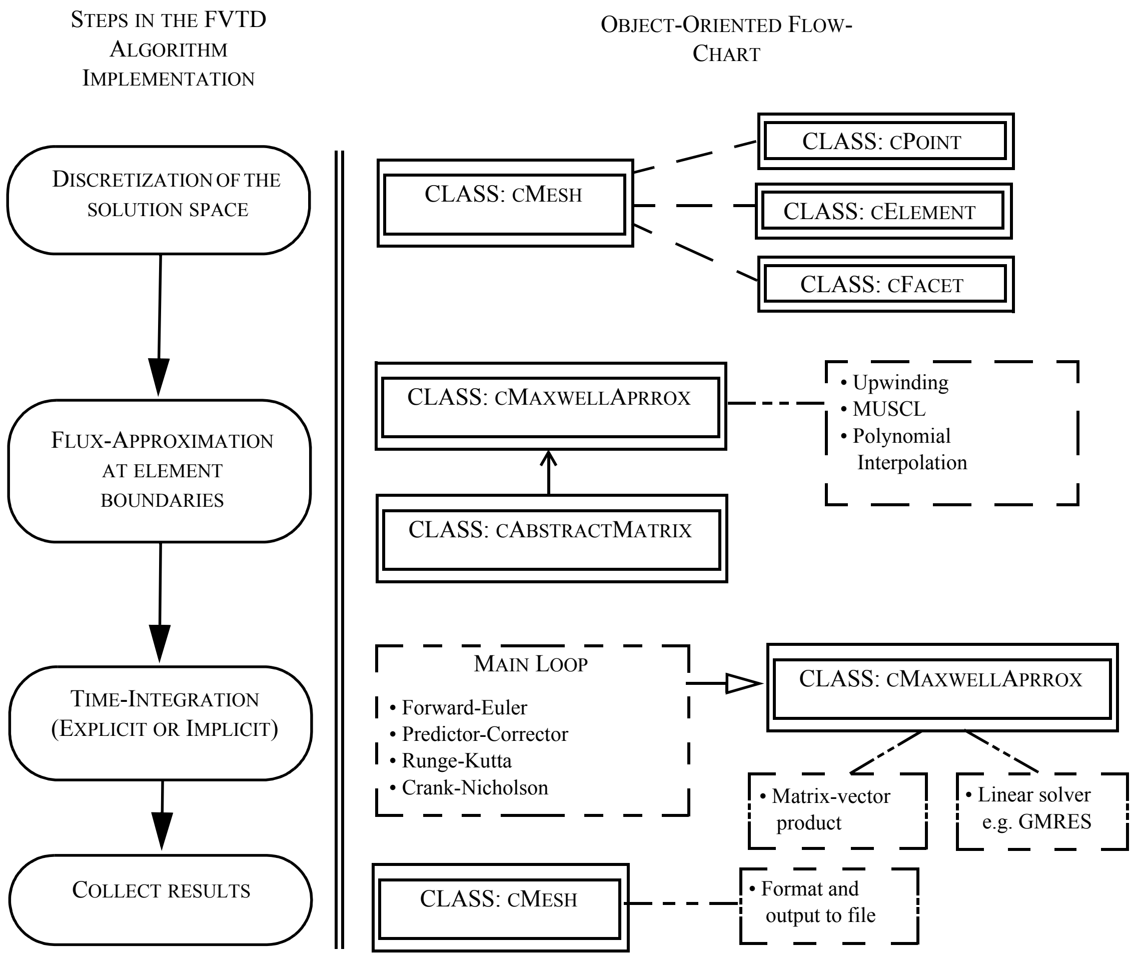
STEPS IN THE FVTD ALGORITHM IMPLEMENTATION
OBJECT-ORIENTED FLOW-CHART
DISCRETIZATION OF THE SOLUTION SPACE
FLUX-APPROXIMATION AT ELEMENT BOUNDARIES
TIME-INTEGRATION (EXPLICIT OR IMPLICIT)
COLLECT RESULTS
CLASS: CMESH
CLASS: CPOINT
CLASS: CELEMENT
CLASS: CFACET
CLASS: CMAXWELLAPRROX
• Upwinding
• MUSCL
• Polynomial Interpolation
CLASS: CABSTRACTMATRIX
MAIN LOOP
• Forward-Euler
• Predictor-Corrector
• Runge-Kutta
• Crank-Nicholson
CLASS: CMAXWELLAPRROX
• Matrix-vector product
• Linear solver e.g. GMRES
CLASS: CMESH
• Format and output to file

Figure 2. Algorithm flow and associated class description

6. Numerical Results: PEC Scattering From Sphere

Herein we present the FVTD results for the back-scattering from a PEC sphere as an exact series solution is available [7]. This problem was selected as a benchmark for the engine as the curved 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. Fig. 3 shows the results for the back-scattering of an *x*-polarized electric-field plane-wave transient pulse varying as the derivative of a Gaussian incident on a three metre radius. The exact solution is presented along with solutions obtained using the MUSCL and third-order polynomial interpolation. We considered a tetrahedral mesh with average element edge length of 0.75 m. The mesh has been truncated using a low-order technique over a spherical boundary. We see that the results are in excellent agreement with the exact solution.

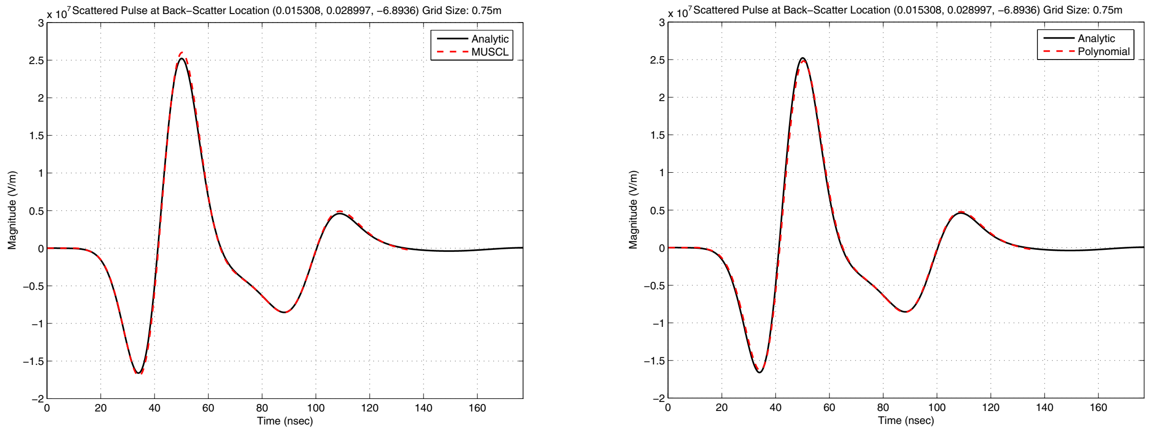

Figure 3. Scattering from a PEC sphere: Back-Scattering for MUSCL (left) and polynomial interpolation (right)

7. Conclusions

The FVTD engine which we have described and implemented, demonstrates the power of the finite-volume time-domain technique for accurately solving Maxwell's equations. Had we not developed the engine in C++ using an object-oriented approach, the investigation of the various flux-interpolation and time-integration schemes would have been much more cumbersome. It was the object-oriented design that allowed us to investigate and compare the different approximation schemes with the final result being the highly-accurate scattering computations shown. In order to obtain similar results using FDTD we required a cubical cell-size having a length of less than 0.1 m.

8. References

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

[2] 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.

[3] 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.

[4] 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.

[5] D. K. Firsov, I. Jeffrey, J. LoVetri, V. Okhmatovski, W. Chamma, "High Order FVTD on Unstructured Grids," submitted to ACES 2006.

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

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