UCRL-ID-151522

Discrete Differential Forms: A Novel Methodology for Robust Computational Electromagnetics

P. Castillo, J. Koning, R. Rieben, M. Stowell, and D. White

January 17, 2003

U.S. Department of Energy

Lawrence
Livermore
National
Laboratory

DISCLAIMER

This work was performed under the auspices of the U. S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48.

Discrete Differential Forms: A Novel Methodology for Robust Computational Electromagnetics

PAUL CASTILLO, JOSEPH KONING, ROBERT RIEBEN, MARK STOWELL, and DANIEL WHITE

Lawrence Livermore National Laboratory

This is the final report for the LLNL LDRD 01-LW-068. The Principle Investigator was Daniel White of the Center for Applied Scientific Computing (CASC). Collaborators included Paul Castillo and Mark Stowell of CASC, and Ph.D students Joe Koning and Rob Rieben of UC Davis. Some of the simulation results in this report were partially funded by a Defense Advanced Research Projects Agency resarch grant, and the two Ph.D. students were supported by the LLNL Student-Employee Graduate Research Fellow program.

We begin with a short Administratve Overview which describes the motivation, scope, and deliverables of this research effort. Then follows the Technical section, which introduces the theory behind our Discrete Diffeential Forms approach, provides an overview of our FEMSTER C++ class library, and concludes with example simulations.

Categories and Subject Descriptors: [**Numerical Analysis**]: Partial Differential Equations; [**Mathematics of computing**]: Mathematical software

General Terms: High order finite element, Computational electromagnetism, Object oriented programming

Additional Key Words and Phrases: $\mathcal{H}(div)$ and $\mathcal{H}(curl)$ conforming finite element methods, Differential forms.

Authors' address: Center for Applied Scientific Computing Research, Lawrence Livermore National Laboratory, Po Box 808 L-551, Livermore CA 94551 white37@llnl.gov.

This work was performed under the auspices of the U.S Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No W-7405-Eng-48.

1. ADMINISTRATIVE OVERVIEW

1.1 Motivation

The goal of this LDRD effort was to research Discrete Differential Forms (DDF), a new methodology for numerical solution of partial differential equations on 3D unstructured grids. This research was motivated by the need for a provably stable, provably conservative, higher-order accurate simulation method for electromagnetics and coupled electro-thermal-mechanical systems. Contrary to popular belief, LLNL's electromagnetic modeling capability is not solely limited by computing hardware, mesh generation, visualization, etc., but is in fact limited by the numerical methods currently employed to solve the equations. The DDF methodology developed under this LDRD effort will be the numerical core of a next-generation unstructured grid electromagnetic code, and will "pave the way" toward a coupled electro-thermal-mechanical modeling capability.

1.2 History

LLNL has long been at the forefront of large-scale scientific computing. Not only does LLNL possess the world's fastest computers, but LLNL simulation codes such as ALE3D (mechanics), Ardra (radiation transport), Jeep (quantum mechanics), and sPPM (fluid/gas dynamics) set the standard for large-scale scientific computing. Unfortunately, Computational Electromagnetics (CEM) at LLNL has not kept pace with the other disciplines. While there exits a plethora of special-purpose, single-use CEM codes, LLNL does not currently have a robust, general purpose, unstructured grid CEM code that is suitable for quantitative engineering analysis. Previous research efforts in time-domain unstructured grid CEM within the Electronics Engineering Department have led to the development of MFV (Madsen 1985-1990), DSI (Madsen, 1990-1997), and TIGER (Steich, 1997-2000) but unfortunately the numerical methods used in these codes have been demonstrated to be numerically unstable for most classes of problems. It can be shown that the instability is not due to the time integration scheme (and hence cannot be fixed by switching to an implicit method) but is due to improper discretization of the curl operator. In addition, the numerical methods used in these codes cannot easily be extended to higher-order approximations, they do not accurately model inhomogeneous materials, and they cannot be directly applied to coupled electro-thermal-mechanical problems. On the other hand, the DDF methodology we developed does not suffer from these limitations.

Table I. Comparison of the numerical properties of the proposed DDF versus the DSI method used in existing CEM codes.

	existing DSI/TIGER codes	proposed DDF
provably stable	NO	YES
provably conservative	NO	YES
higher order	NO	YES
EM boundary conditions	NO	YES
multiphysics capable	NO	YES

Our methodology for CEM is based upon the theory of differential forms. While

the theory of differential forms is mature and is often used in theoretical physics, it has been slow to percolate down to computational engineering. In simple terms, we will develop Discrete Differential Forms (DDF), which are finite element basis functions defined on a mesh. These discrete differential forms mimic the properties of the true continuous differential forms. The basis functions are then used to discretize the partial differential equation yielding a discrete system of equations that can be solved on a computer. The key point is to acknlowledge that different physical quantities (e.g. electric field, magnetic flux, electric current, etc.) have different mathematical properites (contiuity, differentialbility, etc.) and therefore in a finite element simulation code different basis functions must be used for the different fields in order to "get the physics right." Some of these different properties are shown in Table II below.

Table II. Distinguishing characterestics of Discrete Differential Forms. For each form we show the corresponding deriviative and intergral that is associated with the form, as well as the physcical quantities that can be discretized by the given form.

Form	*Field*	*Continuity*	*Derivative*	*Integral*
0-form	electric potential Φ temperature T	gradient	point	total
1-form	electric field E magnetic potential A	curl	line	tangential
2-form	magnetic flux B electric current J	divergence	surface	normnal
3-form	charge density ρ energy density Ψ	N/A	volume	none

Preliminary research has shown that discretizing an E-B formulation of Maxwell's equations using a differential forms based approach yields a provably stable, provably charge and energy conserving discretization of the time-dependent Maxwell's equations. The EMSolve code developed by Daniel White and Joe Koning is an example of a DDF code. This code currently uses four different types of finite element basis functions: the classic nodal basis functions are used for continuous scalar quantities such as the electrostatic potential (a 0-form field), H-curl and H-div vector basis functions are used for the electric field (a 1-form) and the magnetic flux density (a 2-form) respectively, and piecewise constant functions are used for charge density (a 3-form). These basis functions are illustrated in Figure 1 for a tetrahedron. The EMSolve code can be used to solve a variety of electromagnetic problems including electrostatics, magnetostatics, electromagnetic wave propagation, and electromagnetic eigenvalue problems. The EMSolve code provides a provably stable and conservative solution of Maxwell's equations. The primary limitation of this code is that it uses only first-order finite element basis functions, and hence a highly resolved mesh is required for accurate field calculations. Our primary goal of this LDRD effort was to develop higher-order DDF basis functions that will allow for the efficient simulation of electrically large structures, such as photonic devices, without the need for a prohibitively fine mesh. A secondary goal

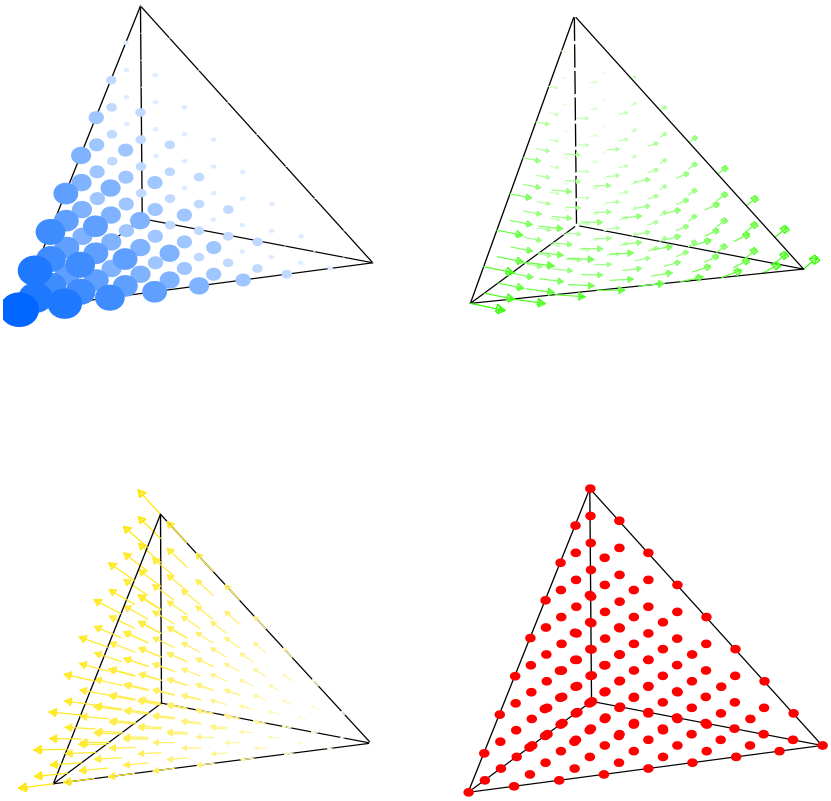

Fig. 1. Lowest order Discrete Differential Forms for a tetrahedron. The upper left function is a 0-form. The upper right function is a 1-form. The lower left function is a 2-form. The lower right function is a 3-form. All four of these basis functions are used, simultaneously, in the EMSolve code.

is research the application of DDF methods to coupled electro-thermal-mechanical problems.

1.3 Scope

To summarize, our goal was to research and develop a Discrete Differential Form (DDF) methodology for unstructured grid computational electromagnetics. This methodology would be more robust than existing methods in the sense that it would be both stable and conservative. In addition it would be higher-order accurate, enabling the efficient simulation of electrically large problems. The three components of our plan were:

I Research higher-order DDF-based finite element basis functions of form 0-

form, 1- form, 2-form, and 3-form for unstructured meshes. The intent is to develop a unified mathematical description of higher-order DDF finite element basis functions and to mathematically analyze, to the extent possible, the numerical properties (stability, conditioning, dispersion error, etc.) of the DDF methodology.

II Develop a software library that encapsulates the DDF-based basis functions and differential operators in a package that can easily be used by application developers. We intend that this library could be openly released to universities for educational and/or research purposes, and perhaps licensed to industry.

III We will seek out interesting electromagnetic and coupled electro-thermal-mechanical problems and use our DDF methodology to solve these problems. This is an important part of our exit plan, as we need to demonstrate the value of our approach on real problems in order to secure follow-on funding. This will be performed in collaboration with LLNL Engineering.

1.4 Conclusion

Our plan was ambitious, but nevertheless we completed tasks I and II above. On the theoretical aspects of our research we collaborated with Prof. Ralf Hiptmair from the Swiss Federal Institute of Technology, Zurich. With Prof. Hiptmair we defined the key characteristics of higher-order discrete differential forms and also constructed very general definitions of degrees-of-freedom and associated transformation rules. We also spent considerable time investigating the conditioning of higher-order basis functions, which resulted in a novel well-conditioned "spectral" element for electromagnetics. Much of this research is contained in the Technical section below.

We completed a software library named FEMSTER, this library contains DDF basis functions, integration rules, and bilinear forms. The library is written in the C++ language and is documented using the Doxygen package. The documentation is on-line at

`http:\\www.llnl.gov\casc\projects\femster.`

The source code for the library has been given to collaborators in LLNL Engineering, at Sandia National Laboratory, at University of Houston, and at Brigham Young University. UC Davis student Rob Rieben will continue to use the FEMSTER library in his Ph.D. research.

We began to validate our higher-order DDF approach by solving canonical problems in electrostatics, electromagnetics, and acoustics. We did in fact verify the predicted higher-order rates of convergence for these canonical problems. Some of these results are shown in the Technical section below. However more work needs to be done in the area of validation and analysis. For example, we have yet to quantify the computational efficiency of employing higher-order basis functions, i.e. while using a higher-order bases significantly reduces the number of unknowns required to obtain a prescribed accuracy, the resulting linear system becomes more difficult to solve. There is likely to be some particular order of approximation that yields an optimal accuracy vs. CPU time "sweet spot". As a second example of further

research, we suggest that the ideal situation is to employ adaptivity so that the computer program itself determines the optimal order of approximation during run time.

Some of the material in this final report was presented at the 2002 IEEE APS Annual Meeting in San Antonio (White and Rieben, "Generalized High Order Interpolatory 1-Form Bases for Computational Electromagnetics"), and at the 2002 SIAM Annual Meeting in Philadelphia (White and Koning, "A Discrete Forms Framework for Wave Equations "). We were also pleased to be invited to present our research at the Special Session on Geometrical Methods for Discrete Electromagnetics at the 2003 IEEE APS Annual Meeting in Columbus. In addition, a more concise version of this report was accepted for publication in the Special Issue on Computational Electromagnetics of the journal Computer Methods in Engineering and Science. In addition, we are preparing a paper to be submitted for publication in the ACM Transactions on Mathematical Software, and a second paper focusing on applications of our methodology toward photonics problems is being written for publication in the Journal of Computational Physics.

2. TECHNICAL INTRODUCTION

The equations of electromagnetics can be simply and elegantly cast in the language of differential geometry, more precisely in terms of differential forms or p-forms [Deschamps 1981], [Baldomir 1986], [Bossavit 1998]. In this geometrical setting, the fundamental conservation laws are not obscured by the details of coordinate system dependent notation; and, the governing equations can be reformulated in a more compact and clear way using well known differential operators of the exterior algebra such as the exterior derivative, the wedge product, and the Hodge star operator, see [Abraham et al. 1996] for an introduction to differential forms. In this context, a natural framework for the modeling of physical quantities is also provided. For example, the electric potentials can be represented by 0-forms; electric and magnetic fields by 1-forms; electric and magnetic fluxes by 2-forms; and, scalar charge density by 3-forms.

In the context of Galerkin approximations, the choice of the finite element space plays a crucial role in the discretization of partial differential equations. For instance, in numerical approximations of the magnetic and electric field intensities, $\mathcal{H}(curl)$ -conforming finite element spaces (or edge elements) are preferred over traditional nodal vector spaces since they eliminate spurious modes in eigenvalue computations; they are able to properly model the jump discontinuity of the field across material discontinuities; and, they enforce tangential continuity of the vector field which corresponds to the exact physical continuity properties of the electric field. Edge elements were introduced in [Nédélec 1980] and [Nédélec 1986] as a generalization of the mixed finite element spaces introduced by P.A. Raviart and J.M. Thomas in [Raviart and Thomas 1977] for $\mathcal{H}(div)$ -conforming methods, for an extensive analysis of several $\mathcal{H}(curl)$ and $\mathcal{H}(div)$ -conforming methods see [Brezzi and Fortin 1991].

Recently, Hiptmair, motivated by the theory of exterior algebra of differential forms, presented a unified framework for the construction of conforming finite element spaces. Remarkably, both $\mathcal{H}(curl)$ and $\mathcal{H}(div)$ conforming finite element spaces and the definition of their degrees of freedom and interpolation operators can be derived within this framework, see [Hiptmair 1999] for more details

Our primary motivation for the development of FEMSTER lies on $\mathcal{H}(curl)$ and $\mathcal{H}(div)$ -conforming finite element discretizations, using the theoretical framework proposed by Hiptmair. In our terminology, a discrete differential p-form is a finite element basis, typically consisting of polynomials. The discrete form is used as the finite element space in the discretization of a vector field. Given a physical law expressed in the language of differential forms, it is quite straightforward to discretize the problem using our class library.

The second focus of this software is on high-order discretization which can reduce the mesh size, memory usage, and CPU time required to achieve a prescribed error tolerance. This is particularly true for electrically large problems due to numerical dispersion. The Galerkin procedure applied to wave equations suffers from numerical dispersion, which means that the computed phase velocity differs from the physical phase velocity and phase error builds up linearly with respect to distance and time. For the popular lowest order edge elements, it is known that the numerical dispersion relation is second order accurate [Warren and Scott

1994], [Warren and Scott 1995], [White 2000]. Second order accuracy may seem adequate, but for an electrically large problem the phase error may be such that the global error is 100 percent, although the local truncation error is quite small. The phenomena of the global error being significantly greater than the local (or optimal) error for a Galerkin solution of wave equations is sometimes referred to as the pollution effect. This has been more precisely explained in [Babuska et al. 1995], [Babuska et al. 1997].

An object-oriented programming (OOP) paradigm provides a natural and straightforward way of achieving our goals in a single computational framework. Our implementation benefits from three OOP concepts: *abstract data types, inheritance, and data encapsulation*. We use abstract base classes as computational devices to represent general mathematical objects such as elements, integration rules, polynomial bases, etc. The definition of abstract interfaces models the functionality of the class at the highest level of abstraction, keeping implementation details to the concrete derived classes. Typical methods included in the interface are evaluation of basis functions and their derivatives, local and global coordinate transformations, and generation of mass and stiffness matrices.

The concept of inheritance avoids the *redevelopment* and *testing* of existing code, by *reusing* base class members (data and methods). Moreover it allows the possibility of *extending* the library by including user-defined data types which can be derived from the existing base classes. Finally, by hiding internal details while providing a public interface, data encapsulation prevents unexpected modifications of data, making the code more robust and modular.

3. PDE'S AND EXTERIOR ALGEBRAS

We begin with the generic boundary value problem stated in the language of differential forms from [Hiptmair 2001]. We assume a 3-dimensional domain Ω with piecewise smooth boundary $\partial\Omega$ partitioned into Γ_D, Γ_N, and Γ_M. The problem statement is

$$du = (-1)^p \sigma, \;\; dj = -\Psi + \Phi \;\; in \;\; \Omega \tag{1}$$

$$T_D u = f \;\; on \;\; \Gamma_D, \;\; T_N j = g \;\; on \;\; \Gamma_N \tag{2}$$

$$j = \star_\alpha \, \sigma, \;\; \Psi = \star_\gamma \, u \;\; in \;\; \Omega \tag{3}$$

$$T_M j = (-1)^p \star^\beta \;\; T_M u \;\; on \;\; \Gamma_M. \tag{4}$$

Here u is a $(p-1)$-form, σ is a p-form, j is a $(3-p)$-form, and both Ψ and Φ are $(3-p+1)$-forms, where $1 \leq p \leq 3$. The variable Φ is a source term. In (1) the operator d is the exterior derivative which maps p-forms to $(p+1)$-forms. In the boundary conditions (2) and (4) the symbol T denotes the trace operator, where the trace of a p-form is an integral over a $p-1$-dimensional manifold. In (3) and (4) the $\star$ symbol denotes the Hodge-star operator, which converts p-forms to $(3-p)$-forms and typically involves material constitutive properties. Equations (1) and (3) can be combined to yield the general second-order elliptic equation

$$(-1)^p d \star_\alpha \;\; du = - \star_\gamma \;\; u + \Phi. \tag{5}$$

The wedge product of differential forms is used in the definition of bilinear forms. The wedge product of a p-form ω and a q-form η is a $(p+q)$-form ζ

$$\omega^p \wedge \eta^q = \zeta^{(p+q)}, p+q \leq 3. \tag{6}$$

If $p+q=3$ then $\omega^p \wedge \eta^q$ is an volumetric energy density like quantity and can be integrated over a volume to yield energy. If $p+q=2$ then $\omega^p \wedge \eta^q$ is a flux density like quantity and can be integrated over a surface to yield net flux.

A Galerkin finite element solution of the generic second-order equation (5) will require bilinear forms. Using the exterior algebra, the bilinear forms required in the Galerkin finite element method can be easily formulated from the general second-order equation (5) by taking the wedge product with an $(l-1)$-form v and integrating over the volume Ω,

$$\int_\Omega (-1)^l d \star_\alpha \;\; du \wedge v = - \int_\Omega \star_\gamma \, u \wedge v + \int_\Omega \Phi \wedge v. \tag{7}$$

Using the integration-by-parts formula

$$\int_\Omega d\omega \wedge \eta + (-1)^l \int_\Omega \omega \wedge d\eta = \int_{\partial\Omega} \omega \wedge \eta \tag{8}$$

yields the two key symmetric bilinear forms

$$a(u,v) \;=\; \int_\Omega \star_\alpha\,(du)\wedge dv, \tag{9}$$

$$b(u,v) \;=\; \int_\Omega \star_\gamma\, u\wedge v. \tag{10}$$

With the definition of the generic Hilbert space

$$H(p)=\left\{u\in\Omega^p : \int_\Omega u\wedge\star u+\int_\Omega du\wedge\star du<\infty\right\}, \tag{11}$$

the Galerkin form of the generic second-order equation (5) can now be expressed as

$$\begin{array}{l} \textit{Given the source function } \Phi \textit{ and the boundary condition } g,\\ \textit{find } u\in\{u\in H(p), T_D(u)=f\}\\ \textit{such that } a(u,v)=b(u,v)+c(u,\Phi)+d(u,g)\\ \textit{for all } v\in\{v\in H(p), T_D(v)=0\} \end{array} \tag{12}$$

where the source term and boundary condition term are given by

$$c(u,\Phi) \;=\; \int_\Omega u\wedge\Phi \tag{13}$$

$$d(u,g) \;=\; \int_{\partial\Omega} \star_\alpha\, du\wedge g. \tag{14}$$

With these generic bilinear forms, source terms, and boundary conditions we can construct a wide variety of model equations that can be solved via the finite element method. The key point is that in order for the finite element procedure to work, it is necessary to use the proper p-form basis functions when discretizing the above bilinear forms.

We are also concerned with time dependent phenomena. The time derivative does not effect the degree of a form. In the diagram below we show the time-dependent Maxwell's equations, where d denotes the spatial derivative and dt denotes the time derivative and converging arrows denote summation. In these diagrams ϕ is the scalar potential 0-form, the 1-forms A, E, and H are the magnetic vector potential, the electric field, and the magnetic field, respectively; the 2-forms B, D, and J are the magnetic flux density, the electric flux density, and the electric current density, respectively; and ρ is the scalar charge density 3-form. The left diagram encompasses Faraday's law $dE - dB/dt = 0$, Coulomb's law for the magnetic field $dB = 0$, and the fact that the electric field E can be written in terms of potentials as $E = d\phi - dA/dt$. The right diagram encompasses Ampere's law $dH - dD/dt = J$, Coulomb's law for the electric field $dD = \rho$, and the continuity equation $dJ - d\rho/dt = 0$. The two diagrams are connected by the constitutive relations $D = \star_\epsilon E$ and $B = \star_\mu H$.

$$
\begin{array}{lccccc}
\text{0-forms :} & & \phi & & & \\
& & \downarrow d & & & \\
\text{1-forms :} & A & \xrightarrow{-dt} & E & & H \\
& \downarrow d & & \downarrow d & & \downarrow d \\
\text{2-forms :} & B & \xrightarrow{-dt} & 0 \quad D & \xrightarrow{-dt} & J \\
& \downarrow d & & \downarrow d & & \downarrow d \\
\text{3-forms :} & 0 & & \rho & \xrightarrow{-dt} & 0
\end{array}
$$

A wave equation can be derived by combining the two diagrams and solving for E,

$$dt\ dt\ \star_\epsilon E = d{\star_\mu}^{-1} dE - dt\ J. \tag{15}$$

This wave equation resembles the generic second order equation (5) with the addition of temporal derivatives, the same bilinear forms (9) and (10) are required for either the elliptic problem or the wave equation.

From the two Maxwell diagrams, Poynting's theorem of energy conservation can be derived by combining the appropriate wedge products and integrating over a volume,

$$\int_\Omega J \wedge E + \int_\Omega E \wedge \star_\epsilon dt E + \int_\Omega H \wedge \star_\epsilon dt H + \int_{\partial\Omega} E \wedge H = 0. \tag{16}$$

The first term is the power deposited in the volume by the independent current source J. The second and third term represent the time rate of change of stored electric and magnetic energy, respectively. In each of the these terms the integrand is a 3-form volumetric power density that can be integrated over a volume. The fourth term represents the net power flowing thorough the surface enclosing the volume, where the integrand is a 2-form flux density known in engineering as the Poynting vector.

4. FEMSTER : A FINITE ELEMENT CLASS LIBRARY

The philosophy of the FEMSTER library is derived from the formulation of an abstract conforming finite element method, see [Ciarlet 1978]. From the implementation point of view, such a formulation is uniquely determined by the 4−tuple $(\Sigma, \mathcal{P}, \mathcal{A}, \mathcal{Q})$ where:

—Σ is a reference element.

—$\mathcal{P}$ is a finite element space defined on Σ .

—$\mathcal{A}$ is the set of degrees of freedom.

—$\mathcal{Q}$ is a quadrature rule defined on Σ .

This abstract formulation can be easily translated into a practical modular code by using an Object-Oriented Programming (OOP) paradigm. The C++ programming language, [Stroustrup 1991], was used in the current implementation. In the following subsections we describe the classes that form the core of the FEMSTER library.

4.1 The class Element3D

The abstract class Element3D describes an interface of a reference element. It consists of a small set of methods that provide all the geometrical information needed in finite element computations. In Table III we present the complete interface with a brief description of each method.

Table III. Interface of the Element3D class

Method	Description
getOrder()	get the order of geometry
getNodes()	get the coordinates of the nodes
setNodes()	set the coordinates of the nodes
jacobian()	get the Jacobian matrix at a point
localToGlobal()	get global coordinates
globalToLocal()	get local coordinates

The library supports the most common types of reference elements: tetrahedron, hexahedron and prism. In Figure 2 we show the inheritance class diagram.

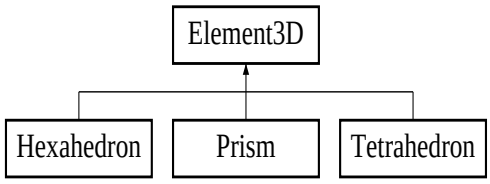

Fig. 2. Element3D class inheritance

The geometry of an arbitrary element is uniquely defined by the physical coordinates of its nodes, its geometrical order and a local mapping that transforms the reference element onto the actual element. These are commonly known in the finite

element literature as sub/iso/- parametric elements, see for example [Ciarlet 1978]. We use the term *vertex* to denote the particular nodes that define the end points of edges and the corners of faces and elements, therefore a tetrahedron always has 4 vertices but may have many more nodes if it is a curved element. We also assume that every node has a unique global integer ID associated with it. All the elements of a same type and having the same geometrical order are topologically equivalent to a single reference element and can be represented through a unique object. Thus only one instantiation of a concrete reference element is needed to represent a block of elements that are topologically equivalent. Geometrical information of a particular element is obtained by, first, setting the coordinates of each node in the reference element, and; then querying this element for the desired information. In Figure 3 we show the topological connectivity standard that we have adopted for our reference elements, this is the required ordering of the vertices assumed by our implementation of the element classes. Each element is defined by a set of generic vertices (implemented as integer IDs). In addition to the connectivity of the reference element, standards must also be adopted for local edge and face orientations on the reference elements. This information is summarized in Tables IV and V.

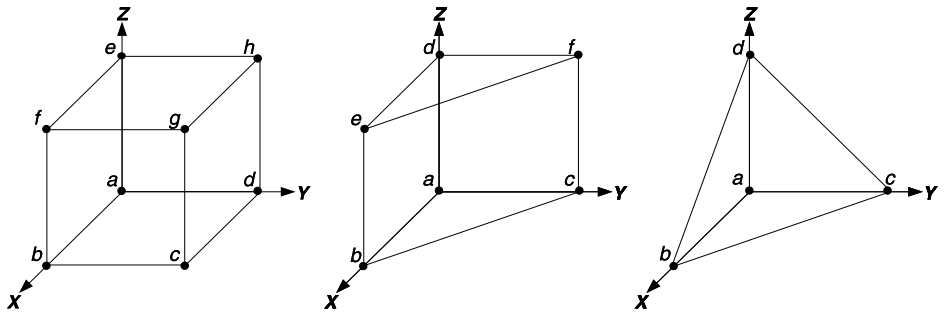

Fig. 3. Topology used to define the reference elements.

In the following lines of code we illustrate how to get the Jacobian matrix of all the elements on a block of linear tetrahedrons.

```
Element3D* element = new Tetrahedron(1);
R3 localPoint = R3(0.0,0.0,0.0);

for (int i = 0; i < numElementInBlock; ++i)
{
  R3xR3 JacMat;
  R3* nodePtr = nodeArray + i*4;
  element->setNodes(nodePtr);
  element->jacobian(localPoint,JacMat);
  ...
```

Table IV. Local edge connectivity for the reference elements

Edge	Hexahedron	Prism	Tetrahedron
1	$\{a, e\}$	$\{a, d\}$	$\{a, b\}$
2	$\{d, h\}$	$\{b, e\}$	$\{a, c\}$
3	$\{b, f\}$	$\{c, f\}$	$\{a, d\}$
4	$\{c, g\}$	$\{a, b\}$	$\{b, c\}$
5	$\{a, d\}$	$\{b, c\}$	$\{b, d\}$
6	$\{e, h\}$	$\{c, a\}$	$\{c, d\}$
7	$\{b, c\}$	$\{d, e\}$	-
8	$\{f, g\}$	$\{e, f\}$	-
9	$\{a, b\}$	$\{f, d\}$	-
10	$\{d, c\}$	-	-
11	$\{e, f\}$	-	-
12	$\{h, g\}$	-	-

Table V. Local face connectivity for the reference elements

Face	Hexahedron	Prism	Tetrahedron
1	$\{a, b, c, d\}$	$\{c, a, d, f\}$	$\{a, b, c\}$
2	$\{e, f, g, h\}$	$\{a, b, e, d\}$	$\{a, b, d\}$
3	$\{a, b, f, e\}$	$\{b, c, f, e\}$	$\{a, c, d\}$
4	$\{d, c, g, h\}$	$\{a, c, b\}$	$\{b, c, d\}$
5	$\{a, d, h, e\}$	$\{d, e, f\}$	-
6	$\{b, c, g, f\}$	-	-

}

4.2 The class IntRule3D

The computation of local integrals arising in stiffness and mass matrices and local load vectors is performed numerically using high order integration rules. The abstract class IntRule3D describes a general interface for an integration rule of an arbitrary three dimensional reference element. In Table VI we present the interface and in Figure 4 the inheritance class diagram.

Table VI. Interface for integration rules in 3D

Method	Description
getNumPts()	get number of quadrature points
getOrder()	get order of exactness
getPoints()	get array of integration points
getWeights()	get array of integration weights
getRegion()	get region tag
getIntegral()	get approximation of the integral

The concrete classes in the bottom of Figure 4 are implementations of high order integration rules for the three types of elements currently available in the library. These are based on tensor products of one dimensional weighted Gauss-Jacobi quadratures. While this process allows to control the degree of exactness of a quadrature rule, in general, it may not produce optimal integration rules, i.e.

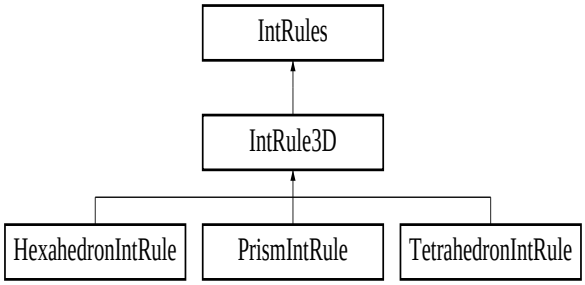

Fig. 4. IntRule3D class inheritance

with a minimum number of points. However the class has been designed to be extensible, the user can provide its own integration rules if desired.

The following lines of code illustrate how to compute the integral of a function f on a single prismatic element using an integration rule which is exact for polynomials of degree less or equal than 2.

```
R3 nodes[] = {...};

Elment3D * element = new Prism(1);
assert( element != 0 );
element->setNodes(nodes);

IntRule3D *    intRulePtr = new PrismIntRule(2);

int            numPoints  = intRule->getNumPts();
const R3 *     point      = intRule->getPoints();
const double * weight     = intRule->getWeights();

double sum = 0.0;
for (int k = 0; k < numPoints; ++k)
{
  R3 x; element->localToGlobal(point[k],x);
  sum += f(x)*weight[k]*element->jacobian(point[k]);
}
```

Observe that by using the abstract interface the same code will perform the same computation on a different type of element. The only necessary changes are the creation of the reference element and its corresponding quadrature.

4.3 The class p-Form

4.3.1 *Polynomial spaces.* The first step in a finite element approximation is to choose the appropriate finite element space $\mathcal{P}$ from which the basis will be constructed. Usually this space consists of a particular set of polynomials. Typical examples are the nodal or Lagrangian polynomial space for the standard H^1 con-

forming method, [Ciarlet 1978]; the edge elements proposed by Nédélec for $\mathcal{H}(curl)$ - conforming methods, [Nédélec 1980; 1986]; and, the face elements proposed by P.A Raviart and J.M Thomas for $\mathcal{H}(div)$ - conforming methods, [Raviart and Thomas 1977]

When implementing the finite element space $\mathcal{P}$ in the context of differential forms, the explicit formulation of the space depends on the p-form and the topology of the reference element Σ . The construction of the finite element space $\mathcal{P}$ is not unique, we choose a construction that leads to a simple and efficient implementation. We use polynomials similar to those described in [Graglia et al. 1997] and [Graglia et al. 1998] as a *primitive basis*. However these are constructed on the reference element Σ rather than in the physical coordinate system. The actual bases used in the finite element procedure are written as a linear combination of the primitive basis. For example, non-uniform interpolatory functions, moment-based functions, orthogonal functions, etc. can all be expressed in terms of the primitive basis.

4.3.2 *Degrees of freedom.* The set $\mathcal{A}$ of degrees of freedom is a finite subset of $\mathcal{P}'$, (i.e. the set of linear functionals from $\mathcal{P}$ onto $\Re$ [Ciarlet 1978]), and satisfies three important properties; namely

—*Unisolvence*: $\{\alpha_i\}$ is dual to the finite element space $\mathcal{P}$; i.e. there exists a set $\{w_j\} \subset \mathcal{P}$ such that $\alpha_i(w_j) = \delta_{i,j}$.
—*Invariance*: degrees of freedom remain unisolvent upon a change of variables; this implies they are not affected by the pullback operation; i.e. $\hat{\alpha}_i \circ \Phi^* = \alpha_i$ (see section 4.4).
—*Locality*: the trace of a basis function on a sub-simplex is determined by degrees of freedom associated *only* with that sub-simplex.

The set $\mathcal{A}$ is defined in terms of moment integrals over sub-simplices of the reference element Σ [Hiptmair 1999]. If we denote a sub-simplex of the reference element Σ of dimension n as Σ_n, then the generalized form for the linear mapping is given by

$$\{\alpha_i\} = \{w \mapsto \int_{\Sigma_n} w \wedge q_n\}, \tag{17}$$

where $p \leq n \leq 3$ and q_n is an $(n-p)$-form *weighting polynomial* of n-variables defined over the sub-simplex Σ_n.

As an example, consider the degrees of freedom for 1-forms. For this case, $p = 1$ and we have the following three sets of moments which will require line integrals over edges weighted by 1-dimensional 0-forms (i.e. 1-dimensional scalar functions), surface integrals over faces weighted by 2 dimensional 1-forms (i.e. vector functions defined in a plane) and volume integrals over the element weighted by 3-dimensional 2-forms (i.e. vector functions defined in a volume).

$$\begin{aligned}
\alpha(\mathbf{w}) &= \int_{\hat{e}} (\mathbf{w} \circ \Phi) \cdot \partial\Phi^T(\vec{\mathbf{t}}q), \\
\alpha(\mathbf{w}) &= \iint_{\hat{f}} (\mathbf{w} \circ \Phi) \cdot \partial\Phi^T(\vec{\mathbf{n}} \times \mathbf{q}), \\
\alpha(\mathbf{w}) &= \iiint_{\vec{v}} (\mathbf{w} \circ \Phi) \cdot \partial\Phi^T \mathbf{q},
\end{aligned}$$

where $\vec{\mathbf{t}}$ denotes the unit tangent vector for each of the edges of Σ and $\vec{\mathbf{n}}$ denotes the unit normal vector for each of the faces of Σ . By appropriately using the iso-parametric mapping Φ and its derivative $\partial\Phi$, we can perform the integrals over the reference element, while maintaining generality of the function $\mathbf{w}$ (i.e. $\mathbf{w}$ can be defined over an arbitrary element).

While this formal definition is sufficiently general to define degrees of freedom of arbitrary type, we find that in practice there are simplifications that can be made to this definition for the particular case of *interpolatory* bases that can significantly improve computational performance. In this case, the integral forms of the degrees of freedom are replaced with simple point evaluation operations. For example, the scalar valued 0-form interpolatory bases can have their degrees of freedom reduced to the familiar form

$$\{\alpha_i(w)\} = w(\Phi(x_i)), \tag{18}$$

where x_i is a particular interpolation point in the reference coordinate system. Similarly, the vector valued 1-form interpolatory bases can have their degrees of freedom reduced to the form

$$\{\alpha_i(\mathbf{w})\} = \mathbf{w}(\Phi(x_i)) \cdot \partial\Phi^T(\vec{\mathbf{t}}_i), \tag{19}$$

where x_i is a particular interpolation point in the reference coordinate system and $\vec{\mathbf{t}}_i$ is an "interpolation vector" associated with the point x_i. Table VII summarizes these simplified linear functionals for interpolatory degrees of freedom.

Table VII. Simplified Degrees of Freedom for Interpolatory Bases

Form	Linear Functional
0-forms	$\{\alpha_i(w)\} = w(\Phi(x_i))$
1-forms	$\{\alpha_i(\mathbf{w})\} = \mathbf{w}(\Phi(x_i)) \cdot \partial\Phi^T(\vec{\mathbf{t}}_i)$
2-forms	$\{\alpha_i(\mathbf{w})\} = \mathbf{w}(\Phi(x_i)) \cdot \lvert\partial\Phi\rvert\partial\Phi^{-1}(\vec{\mathbf{n}}_i)$
3-forms	$\{\alpha_i(w)\} = \lvert\partial\Phi\rvert w(\Phi(x_i))$

Using this general approach, we can construct a discrete differential p-form basis of order k in the following manner. We begin by generating a primitive basis $W = \{w_j\}$. This primitive basis can be made in a number of different ways depending on the degree of the form and topology of the element. For example, a primitive basis on a reference tetrahedral element can be made using the interpolatory procedure of [Graglia et al. 1997], while for a reference hexahedron, we can form a primitive basis by taking tensor direct products of 1-dimensional Lagrange interpolatory polynomials. In order to construct a new basis (non-uniform interpolation, hierarchical, etc.) from the primitive basis we first formulate the linear functionals for $\mathcal{A}$ using the appropriate weighting polynomials. The choice of weighting polynomials in the formal definition of the degrees of freedom will determine the type of the new basis. For example, by choosing orthogonal weighting polynomials, the new basis will be hierarchical. We then apply the projection operation to the primitive basis; i.e. we construct the matrix

$$V_{i,j} = \alpha_i(w_j); \quad w_j \in W \tag{20}$$

This system, which is similar to a Vandermonde matrix, is a linear mapping which expresses the new basis in terms of the primitive basis and will have a rank equal to the dimension of the primitive basis. We know from the definition of the degrees of freedom that the unisolvence property must hold for the new basis; so in order to satisfy this requirement, we must find the inverse of the Vandermonde matrix. The newly defined basis, which we will denote as F will then have the form:

$$F = V^{-1}W \tag{21}$$

Therefore, the newly defined basis will be a linear combination of the primitive basis that spans the space $\mathcal{P}$ (i.e. satisfies the unisolvence property). While it may seem computationally expensive to invert a matrix in order to get the new basis, it should be noted that this is a one time cost as the inverse of the Vandermonde matrix can be stored and used over and over as necessary.

4.3.3 *p-Form class interface.* We have a class hierarchy for each of the p-form bases, the hierarchy for the 0-form class is shown in Figure 5. Concrete classes are presented in the lowest level of the tree. The other p-forms have a similar inheritance diagram. Our Silvester-Lagrange (SL) bases are similar to the bases defined in [Graglia et al. 1997] which use equidistant and shifted equidistant interpolation points. The difference between our SL bases and the bases proposed in [Graglia et al. 1997] is that ours satisfy the properties in Table VII. The uniformly spaced interpolatory bases are suitable for low order approximations, *i.e.*, $k = 1$ to 4. It is well known that this particular choice of interpolation points produce badly conditioned mass and stiffness matrices when high order approximations are used. For this reason we have implemented spectral classes that use arbitrary sets of interpolation points, typically Gauss-Lobatto or Tchebyschev points. As an example, figure 6 shows the number of iterations required for a conjugate gradient algorithm to solve the linear system (with an error tolerance of 10^{-12}) arising from the discretization of Poisson's equation using a 0-form basis on a hexahedral mesh (see section 5.1). In this example we show the results for three different types of interpolatory bases. Note that the results for the SL basis show exponential growth of condition number as the approximation order is increased while the two spectral bases show logarithmic growth. The user can also experiment by passing their own set of interpolation points into the constructor.

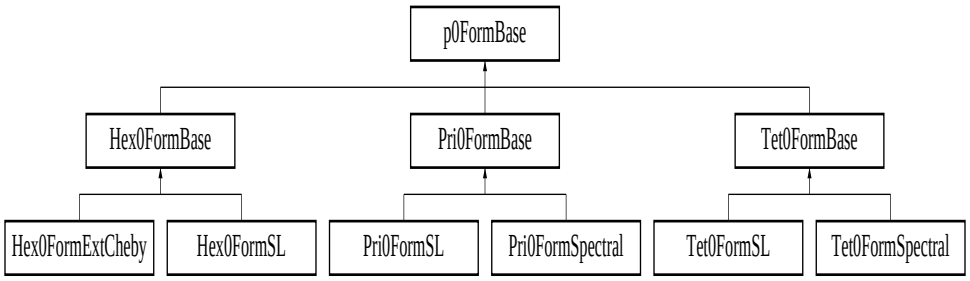

Fig. 5. 0-Form class inheritance

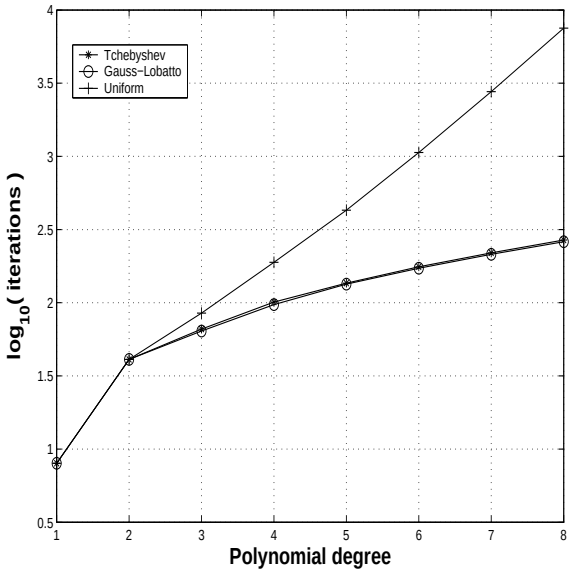

Fig. 6. Iteration count for conjugate gradient solution of Poisson's equation using three different interpolatory bases.

The interface of a p-form includes methods that can be used in the computation of the local matrices and vectors, such as methods to evaluate basis functions and their derivatives at an arbitrary point, to evaluate the interpolate and its derivative and to compute the expansion of the interpolate of a given function. In Table VIII we show part of the interface of a p-form class.

Table VIII. Interface of a p-form class

Method	Description
getOrder()	get the order of the p-form
getDim()	get the dimension of the p-form
setElement()	set the element pointer
clearElement()	clear the element pointer
getConnectivity()	get the connectivity
localEvaluate()	$\phi_i(x), \quad i = 1, \ldots, n$
localEvaluateD()	$d\phi_i(x), \quad i = 1, \ldots, n$
localInterp()	$\Pi(f)(x)$
localInterpD()	$d\Pi(f)(x)$
project()	α_i where $\Pi(f) = \Sigma\alpha_i\phi_i$

The method localEvaluateD() computes the action of a differential operator on the basis functions at a given point. This operator is the exterior derivative and is uniquely determined by the p-form, see [Abraham et al. 1996], [Burke 1985] for a classical geometrical approach. In particular, this operator refers to the gradient for 0-forms; the curl for the 1-forms; and finally, the divergence for the 2-forms.

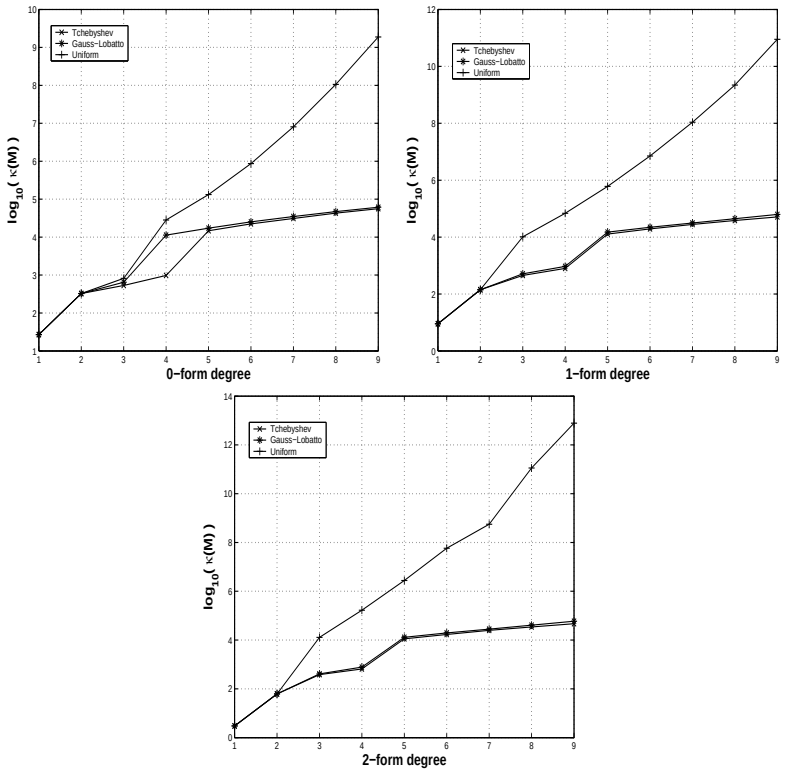

Fig. 7. Condition number of p-form mass matrix using three different sets of interpolation points.

To facilitate the assembly of global mass and stiffness matrices, the basis functions are locally sorted in the following order : nodal basis functions, edge basis functions, face basis functions, and interior basis functions. The getConnectivity() method returns the number of basis functions per vertex, per edge, per face, and per cell.

4.4 Bilinear forms

The purpose of this class is to provide an interface to compute mass and stiffness matrices as well as load vectors. We present the derivation of the global stiffness matrix using the exterior algebra of differential forms.

Let $\mathcal{T}_h$ be a triangulation of a physical domain Ω using tetrahedral, hexahedral, or prismatic elements. We consider the general bilinear form $a(\cdot,\cdot)$ defined by

$$a(u,v) \;=\; \int_{\Omega} *_{\alpha}(du) \wedge dv, \tag{22}$$

where $*_{\alpha}$ is the Hodge operator associated to a symmetric definite positive tensor α, which typically represents material properties such as electric and magnetic permeabilities and conductivities; and, u and v are both p-forms. Then by using the properties of the Hodge operator and the local change of variables given by the iso-parametric mapping $\Phi(\hat{T}) = T$, we re-write the bilinear $a(\cdot,\cdot)$ from equation

(22) as follows

$$a(u,v) = \int_{\Omega} *_{\alpha}(du) \wedge dv \tag{23}$$

$$= \sum_{T \in \mathcal{T}_h} \int_{T=\Phi(\hat{T})} *_{\alpha}(du) \wedge dv \tag{24}$$

$$= \sum_{T \in \mathcal{T}_h} \int_{\hat{T}} \Phi^* \left(*_{\alpha}(du) \wedge dv\right) |\Phi| \tag{25}$$

$$= \sum_{T \in \mathcal{T}_h} \int_{\hat{T}} *_{\alpha \cdot \Phi} \left(\Phi^*(du)\right) \wedge \Phi^*(dv) |\Phi|. \tag{26}$$

Similarly the mass matrix can be obtained using the following bilinear form

$$b(u,v) = \int_{\Omega} *(u) \wedge v, \tag{27}$$

and after some manipulations, we have

$$b(u,v) = \sum_{T \in \mathcal{T}_h} \int_{\hat{T}} * \left(\Phi^*(u)\right) \wedge \Phi^*(v) |\Phi|. \tag{28}$$

Equations (26) and (28) show that all calculations for the mass and stiffness matrices are performed on a standard reference element $\hat{T}$ (i.e. the unit cube, tetrahedron, or prism). Results are then transformed to physical mesh elements (of arbitrary curvature) via a set of well defined transformation rules based on the properties of differential forms. These rules are summarized in Table IX. Given these transformations the bases need only be evaluated on the reference element and transformed accordingly. This gives rise to a very computationally efficient algorithm for computing finite element approximations. For a given element topology and basis order, the basis functions only need to be computed once. Then, for every element of the same topology in the mesh, the results from the reference element can simply be mapped according to the transformation rules. This can significantly reduce computational time for a typical finite element computation. In addition, integration over the reference element is much simpler and can be done exactly using Gaussian quadrature of the appropriate order.

Table IX. Transformation rules Φ^*

	$\Phi^*(u)$	$\Phi^*(du)$
0-forms	$u \circ \Phi$	$\partial\Phi^{-1}(du \circ \Phi)$
1-forms	$\partial\Phi^{-1}(u \circ \Phi)$	$\frac{1}{\lvert\partial\Phi\rvert}\partial\Phi^T(du \circ \Phi)$
2-forms	$\frac{1}{\lvert\partial\Phi\rvert}\partial\Phi^T(u \circ \Phi)$	$\frac{1}{\lvert\partial\Phi\rvert}(du \circ \Phi)$

In addition, several levels of efficiency have been added in the implementation of this class. The local mass and stiffness matrices are symmetric therefore only one triangular block is actually computed and the rest of the entries are copied. For tetrahedrons of order 1, the Jacobian is constant so there is no need to compute it at each integration point.

Below, in Table X, we show the common interface of a bilinear p-form.

Table X. Interface of the Bilinear-pForm class

Method	Description
setpForm()	set a specific 0-form
setIntRule()	set the integration rule
setElement()	set a specific element3D
initialize()	initialize internal data
getMassMatrix()	get the local mass matrix
getStiffnessMatrix()	get the local stiffness matrix
getLoadVector()	get local load vector
getUError()	get the local error
getQError()	get local error of derivative

4.5 Permutations

When assembling a global mass or stiffness matrix, it is imperative that all elements which share a sub-simplex agree on the ordering and possibly direction of the degrees of freedom associated with that sub-simplex. One approach is to compute the basis functions directly on the actual element, for every element in the mesh. We disregarded this approach as it is extremely inefficient for higher-order bases. Instead, our basis functions are computed locally at the quadrature points of a reference element and then transformed as described in Section 4.4. In addition to this geometrical transformation it is necessary to perform a permutation (re-ordering) of the basis functions prior to global assembly. The purpose of this permutation is to guarantee that there exists a unique, global definition of the i-th basis function on a given edge or face. The details of this permutations are different for each element type and for each form, hence the details are implemented in different concrete classes as illustrated in Figure 8.

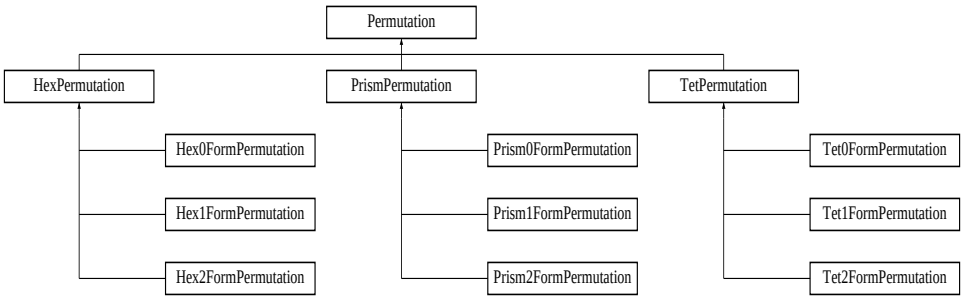

Fig. 8. Permutation class diagram

The Permutation class is an abstract interface designed to allow the user to take local arrays and matrices, computed by the BilinearForms class, and reorder their contents to conform to a global standard. Our implementation of the Permutation class works only for our assumed global standard, if a client program requires some other global standard then a new concrete Permutation class needs to be derived

for this particular standard. We have adopted a standard ordering procedure for the degrees of freedom on edges and faces that is based on the global integer IDs of the vertices that define the edge or face. Our standard for edges is that an edge is directed from low global vertex ID to high global vertex ID, this global orientation is independent of local ordering of element vertices. Our standard for faces is to define a $u-v$ coordinate system with the origin at the face vertex with lowest global vertex ID, the u axis is the edge connecting the origin to the next lowest global vertex ID, and the v axis is the other edge associated with the origin. The face normal is in the $u \times v$ direction. This gives every face in the mesh an global orientation that is independent of local ordering of element vertices. To summarize, the BilinearForm class computes local mass/stiffness matrices and load vectors according to a local element point of view, the Permutation class performs rotations and reflections on these local matrices and vectors to permute the contents to the global standard. The global interface of the permutation class is shown in Table XI.

Table XI. Interface of the Permutations class

Method	Description
createElementPermutation()	create permutation for a given element
permuteVector()	apply permutation to a vector
permuteMatrix()	apply permutation to a matrix
getNumDofPerEdge()	number of dof in edge
getNumDofPerFace()	number of dof in face
getNumDofPerCell()	number of dof in cell
getEdgeDofArray()	get index array of dof in edge
getFaceDofArray()	get index array of dof in face
getCellDofArray()	get index array of dof in cell

In general, an element can have degrees of freedom associated with its vertices, edges, faces and interior. Of these, only the first three are ever shared between two elements, and the sharing of vertex degrees of freedom is trivial. Therefore we need only concern ourselves with the possible reorientation of edge and face based degrees of freedom. We simply enumerate the possible permutations that can be applied, and given the specific global vertex ID's apply the proper permutations. For edges, there are only two cases to consider: the edge is either reversed, or it remains the same. For faces, we employ the symmetry group of a general n-dimensional polygon. For our library, this implies the consideration of both triangular and quadrilateral faces. In general, an n dimensional polygon will have $2n$ symmetry operations, n reflections and n rotations, and this is precisely what is implemented in our concrete Permutations classes These cases are enumerated in Tables XII and XIII.

As an example, consider a 1-form basis of order 3 on a hexahedral mesh. There will be 12 degrees of freedom on each face of the mesh, and we need as global standard as to the precise location and direction of the i-th interpolatory degree of freedom on every face. We have two hexahedral elements that share a face, the shared face is defined by the global IDs 2,5,8, and 11. We define a global orientation of this shared face as follows: begin with the *lowest* integer ID in the face, then traverse cyclically through the list by proceeding to the lowest neighbor of the

Table XII. Symmetry operations for triangular faces

Operation	Description
R0	Rotation of 0 degrees (Identity)
R120	Rotation of 120 degrees
R240	Rotation of 240 degrees
V	Reflection about vertical axis
D1	Reflection about main diagonal
D2	Reflection about second diagonal

Table XIII. Symmetry operations for quadrilateral faces

Operation	Description
R0	Rotation of 0 degrees (Identity)
R90	Rotation of 90 degrees
R180	Rotation of 180 degrees
R270	Rotation of 270 degrees
V	Reflection about vertical axis
H	Reflection about horizontal axis
D1	Reflection about main diagonal
D2	Reflection about second diagonal

lowest integer ID. In this example, the global orientation for the shared face will be 2,5,11,8. This defines the u and v directions of the 1-form degrees of freedom on the face, as well as the ordering of the degrees of freedom. Figure 9 gives a visual example of the required permutations. In this figure the letters *a,b,c,...* denote the ordering of the basis functions as viewed from the left element, from the global standard, and the right element.

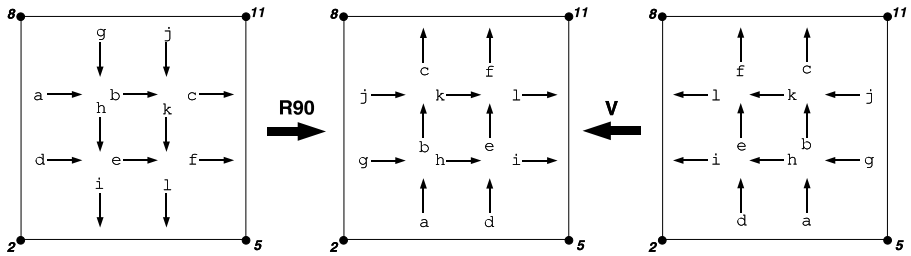

Fig. 9. Example of facial permutation. The center face is the global standard ordering of the 1-form degrees of freedom, determined solely by the global vertex ID's of the face. The other faces show the degrees of freedom, from a local point of view, as viewed by the two elements that share this face.

In addition to reordering local arrays and matrices, the permutation class handles the local integer IDs for the degrees of freedom associated with the sub-simplex of an element. This information is obtained by the member functions getEdgeDofArray(), getFaceDofArray() and getCellDofArray(). These methods proved the user with an integer array containing the local integer IDs for the particular sub-simplex that is queried. This information is useful when applying boundary conditions to the

sub-simplices of mesh elements. The following piece of code shows the usage of such methods in the setting of Dirichlet boundary conditions. The goal is to apply a functional value to the degrees of freedom associated with the bounding surface of a mesh. Since the mesh is 3-dimensional, the bounding surface is referenced by the surface faces of mesh elements. In general, a face can have degrees of freedom on its vertices, along its edges and internal to the face itself.

```
// ------------------------------------------------------------
//                       Boundary conditions
// ------------------------------------------------------------

for (int i = 0; i < mesh->getNumBoundaryFaces(); ++i)
{
  const BFaceInfo & info = mesh->getBoundaryFaceInfo(i);
  R3 nodes[8]; // enough nodes for all the element types

  int cellId = info.cellId_;
  mesh->getElement(cellId,nodes);
  element->setNodes(nodes);

  pForm->project(poisson_u,localVector);

  const int* nodeConnectivity = mesh->getNodeConnectivity(cellId);
  permutation->createElementPermutation(nodeConnectivity);
  permutation->permuteVector(localVector);
  mesh->getMapping(cellId,
                   numDofPerNode,
                   numDofPerEdge,
                   numDofPerQFace,
                   numDofPerTFace,
                   numDofPerCell,
                   mapping);

  // ----------------------------------------------------------
  //           Process dof associated with face nodes
  // ----------------------------------------------------------

  int numNodes = info.numNodes_;
  for (int k = 0; k < numNodes; ++k)
  {
    int L = info.nodeIds_[k];
    applyDirichlet(A,b,x,mapping[L],localVector[L]);
  }

  // ----------------------------------------------------------
  //           Process dof associated with face edges
  // ----------------------------------------------------------
```

```
  int numEdgeDof = permutation->getNumDofPerEdge(info.localId_);

  if ( numEdgeDof > 0 )
  {
    int* edgeDofArray = new int[numEdgeDof];
    assert( edgeDofArray != 0 );
    int numEdges = info.numNodes_;
    for (int k = 0; k < numEdges; ++k)
    {
      permutation->getEdgeDofArray(info.edgeIds_[k],edgeDofArray);
      for (int l = 0; l < numEdgeDof; ++l)
      {
        int L = edgeDofArray[l];
        applyDirichlet(A,b,x,mapping[L],localVector[L]);
      }
    }

    delete [] edgeDofArray;
  }

  // ------------------------------------------------------------
  //           Process dof associated with face interior
  // ------------------------------------------------------------

  int numFaceDof = permutation->getNumDofPerFace(info.localId_);

  if ( numFaceDof > 0 )
  {
    int* faceDofArray = new int[numFaceDof];
    assert( faceDofArray != 0 );

    permutation->getFaceDofArray(info.localId_,faceDofArray);
    for (int k = 0; k < numFaceDof; ++k)
    {
      int L = faceDofArray[k];
      applyDirichlet(A,b,x,mapping[L],localVector[L]);
    }

    delete [] faceDofArray;
  }
}
```

4.6 Poisson driver

The following sample code illustrate a driver for assembling the global stiffness matrix and load vector for the following Poisson problem

$$-\nabla\left(\sigma\nabla u\right) \;=\; f,$$

where σ is a local constant diffusion tensor. We assume the mesh consists of linear tetrahedral elements. For this problem we use a discrete 0-form of degree 3 with a set of equidistant interpolatory points. To make the exposition clear, we have omitted all the details not related to the mathematical aspects of the finite element method.

```
// ------------------------------------------------------------------
//                    CREATE FINITE ELEMENT OBJECT
// ------------------------------------------------------------------

int degree = 3;

Element3D *     element = new Tetrahedron(1);
p0FormBase *    pForm   = new Tet0FormSL(degree);
IntRule3D *     intRule = new TetrahedronIntRule(2*degree+1);
Permutation *   perm    = new Tet0FormPermutation((Tet0FormBase *) pForm);
Bilinear0Form * fem     = new Bilinear0Form;

fem->setIntRule(intRule);
fem->setpForm(pForm);
fem->setElement(element);
fem->initialize();

pForm->setElement(element);

// ------------------------------------------------------------------
//                    INITIALIZE SOME VARIABLES
// ------------------------------------------------------------------

int dim = pForm->getDim();

int numDofPerNode = 0;
int numDofPerEdge = 0;
int numDofPerFace[6];
int numDofPerCell = 0;

pForm->getConnectivity(numDofPerNode,
                       numDofPerEdge,
                       numDofPerFace,
                       numDofPerCell);

int numDofPerQFace = 0;
```

```
int numDofPerTFace = numDofPerFace[0];

int numNodes  = mesh->getNumNodes();
int numEdges  = mesh->getNumEdges();
int numQFaces = mesh->getNumQFaces();
int numTFaces = mesh->getNumTFaces();
int numFaces  = numQFaces + numTFaces;
int numCells  = mesh->getNumCells();

int numDofs = numNodes*numDofPerNode +
              numEdges*numDofPerEdge +
              numQFaces*numDofPerQFace +
              numTFaces*numDofPerTFace +
              numCells*numDofPerCell;

// -------------------------------------------------------------
//                      ALLOCATE LOCAL BUFFERS
// -------------------------------------------------------------

int *    Map = new int[dim];        assert( Map != 0 );
double * Vec = new double[dim];     assert( Vec != 0 );
double * Mat = new double[dim*dim]; assert( Mat != 0 );

// -------------------------------------------------------------
//           ASSEMBLE STIFFNESS MATRIX AND RIGHT HAND SIDE
// -------------------------------------------------------------

CSRmat A; A.beginAssembly(numDofs,numDofs);

for (int cellId = 0; cellId < numCells; ++cellId)
{
  R3 nodes[4];
  mesh->getElement(cellId,nodes);
  element->setNodes(nodes);

  fem->getLoadVector(ffunction,Vec);
  fem->getStiffnessMatrix(D[cellId],Mat);

  const int * nodeConnectivity = mesh->getNodeConnectivity(cellId);
  perm->createElementPermutation(nodeConnectivity);
  perm->permuteVector(Vec);
  perm->permuteMatrix(Mat);
  mesh->getMapping(cellId,
                   numDofPerNode,
                   numDofPerEdge,
                   numDofPerQFace,
```

```
                    numDofPerTFace,
                    numDofPerCell,
                    Map);

  for (int k = 0; k < dim; ++k) b[Map[k]] += Vec[k];
  A.addSubBlock(dim,Map,Mat);
}
```

It is important to observe that after performing a local permutation of the local stiffness matrix and load vector, we still need a mapping that relates local degrees of freedom with global degrees of freedom. This mapping is obtained from our 3D mesh class. As in the local ordering, the global ordering scheme lists the degrees of freedom associated with the vertices first, followed by those associated with the edges, and so on. However, a different mapping scheme could be used.

4.7 Computation of errors

The computation of errors can be a very useful tool for debugging purposes. They can be used to determine the accuracy of the finite element approximation whenever the exact solution of the problem is known; or to validate the numerical rates of convergence with those predicted by theoretical estimates. We have included the methods getUError() and getQError() in the bilinear form interface to compute the local error of the variable and its exterior derivative. These errors are computed in the L_2 norm.

The following code illustrates the use of this methods. We assume that the linear system has already been solved and that the approximated solution is contained in x. Observe that after gathering the degrees of freedom of a particular element in a local vector, we need to apply the inverse of the permutation and then compute the errors.

```
  // ------------------------------------------------------------
  //       COMPUTE APPROXIMATION ERRORS IN THE L2 NORM
  // ------------------------------------------------------------

double errorPotential = 0.0;
double errorGradient  = 0.0;
for (int cellId = 0; cellId < numCells; ++cellId)
{
  R3 nodes[4];

  mesh->getElement(cellId,nodes);
  element->setNodes(nodes);

   // ------------------------------------------------------------
   //   Compute contribution of element i to the global error
   // ------------------------------------------------------------

  const int* nodeConnectivity = mesh->getNodeConnectivity(cellId);
```

```
  permutation->createElementPermutation(nodeConnectivity);
  mesh->getMapping(cellId,
                   numDofPerNode,
                   numDofPerEdge,
                   numDofPerQFace,
                   numDofPerTFace,
                   numDofPerCell,
                   mapping);

  for (int k = 0; k < dim; ++k) localVector[k] = x[mapping[k]];
  permutation->permuteVector(localVector,INVERSE_PERMUTATION);

  errorPotential += fem->getUError(ufunction,localVector);
  errorGradient  += fem->getQError(dufunction,localVector);
}

cout << "Errors = " << sqrt(errorPotential) << " "
     << sqrt(errorGradient) <<  endl;
```

5. VALIDATION

In this section we perform several computational experiments to verify the theoretically predicted rates of convergence. Since we compare computed solutions to exact solutions, by necessity the geometry must be trivial. Only a few of the many validation tests we performed are presented below.

5.1 Poisson Equation

The Poisson equation corresponds to the case $p = 1$ in (1)-(4). Here u is a 0-form potential-like quantity and j is a 2-form flux-like quantity. The operator $\star_\alpha$ can be interpreted as the dielectric constant in the case of electrostatics, permeability in the case of magnetostatics, thermal conductivity in the case of heat transfer, etc. The boundary conditions on Γ_D, Γ_N, and Γ_M correspond to the standard Dirichlet, Neumann, and Robbins boundary conditions, respectively.

$$\begin{aligned} -\nabla\alpha\cdot(\nabla\phi) &= f \quad \text{in } \Omega, \\ \phi &= s \quad \text{on } \partial\Omega_D, \\ \nabla\phi\cdot\hat{n} &= n \quad \text{on } \partial\Omega_N. \end{aligned} \tag{29}$$

In this numerical example we solve the above problem on a cubic domain using hexahedral elements subject to the Dirichlet boundary condition. We choose an exact solution

$$\phi = \cos(x)\sin(y)\exp(z) \tag{30}$$

and insert this into (29) to form the corresponding source function f. We use Bilinear0Form methods getStiffnessMatrix() and getLoadVector() to form the local stiffness matrix and the local load vector for every element in the mesh. Given these local matrices and local vectors, the standard finite element procedure is used to assemble a global system of the form

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{31}$$

where $\mathbf{A}$ is the global stiffness matrix, $\mathbf{b}$ is the global load vector, and $\mathbf{x}$ is the unknown vector of finite element coefficients. The linear system is solved via a conjugate gradient algorithm.

In Figure 10 we show the computed L_2 error versus element size h on a $\log - \log$ scale for 0-form basis functions of degree 1 through 4. The slopes of the lines (based on a least-squares fit of the data points) are $(2.0000, 2.9939, 3.9914, 4.9692)$ indicating the optimal convergence rate of $k + 1$.

5.2 Vector Helmholtz

The vector Helmholtz equation corresponds to the case $p = 2$ in (1)-(4) with the exception that we add a $-\omega^2$ term so that the problem corresponds to a hyperbolic wave equation. Here both u and j are 1-form field-like quantities and σ, ψ, and Φ are 2-form flux-like quantities. In the case of Maxwell's equations u and j are the electric and magnetic fields, Ψ and σ are the electric and magnetic flux densities, the source term Φ is the current density, and the star operators $\star_\alpha$ and $\star_\gamma$ correspond

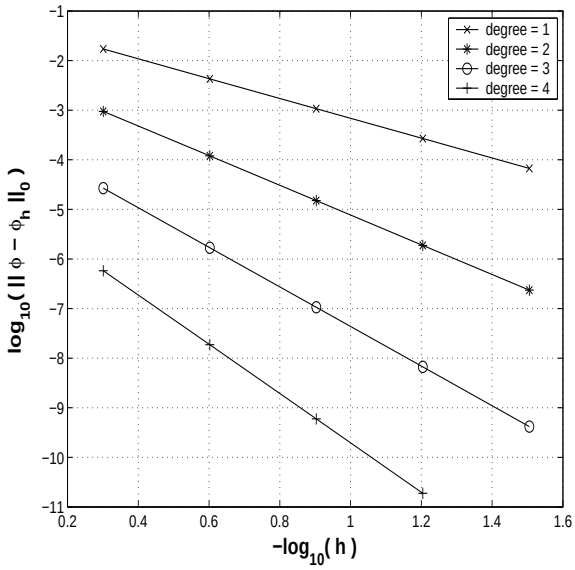

Fig. 10. Polynomial convergence of h-refined solutions of the Poisson equation using finite element 0-form basis functions of degree 1 through 4.

to $1/\mu$ and ϵ, respectively. The boundary conditions on Γ_D and Γ_N correspond to $\vec{n} \times u = 0$ and $\vec{n} \times \nabla \times u = 0$, and the boundary condition on Γ_M is an impedance boundary condition. In standard form we have

$$\nabla \times \frac{1}{\mu} \nabla \times \vec{E} - \epsilon \omega^2 \vec{E} = \vec{f}, \tag{32}$$

where f is a source term consisting of electric or magnetic currents, and $\vec{E}$ is the time-harmonic complex-valued electric field. In practice we use a prescribed voltage boundary condition $\vec{n} \times \vec{E} = v(t)$ or a radiation boundary condition $\vec{n} \times \nabla \times \vec{E} = P(E)$ where $P(E)$ is chosen to approximate the Sommerfeld radiation boundary condition.

In this computational experiment we validate the expected rates of convergence for h-refinement by choosing a simple problem with a known, smooth solution. The computational domain is a unit cube, discretized via a series of unstructured tetrahedral meshes. We choose an exact solution

$$\vec{E} = \left(0, 0, \left(x - x^2\right)^2 \left(y - y^2\right)^2 \left(z - z^2\right)^2\right), \tag{33}$$

and insert this into (32) to form the corresponding source function $\vec{f}$. We use Bilinear1Form methods to form the local matrices and the local load vector for every element in the mesh. Given these local matrices and local vectors, the standard finite element procedure is used to assemble a global system of the form

$$\left(\mathbf{A} - \omega^2 \mathbf{B}\right) \mathbf{x} \;=\; \mathbf{b}, \tag{34}$$

where $\mathbf{A}$ is the global stiffness matrix, $\mathbf{B}$ is the global mass matrix, $\mathbf{b}$ is the global load vector, and $\mathbf{x}$ is the unknown vector of finite element coefficients. The linear system is solved via an ILU preconditioned GMRES algorithm. The ILU preconditioned GMRES code is not part of the FEMSTER distribution.

In Figure 11 we show the computed L_2 error versus element size h on a $\log - \log$ scale for 1-form basis functions of degree 1 through 6. The slopes of the lines (based on least-squares fit of the last three data points) are $(0.98, 1.97, 2.97, 3.97, 4.97, 5.98)$ indicating the optimal convergence. It is interesting to note that for this particular problem using a 6th order basis on a 1440 element mesh yields a solution accurate to 10 significant digits, where a comparable solution using a 1st order basis would require a mesh consisting of billions of elements. Naturally, we cannot expect this type of accuracy for problems with re-entrant corners and associated field singularities, but high-order approximation can be combined with adaptive h-refinement for such problems.

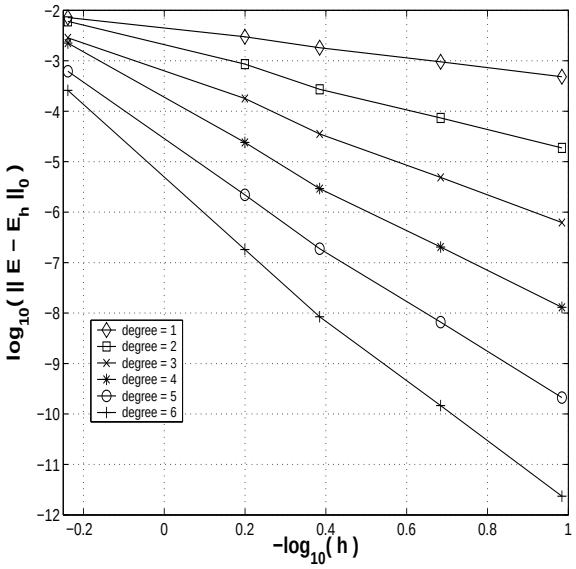

Fig. 11. Polynomial convergence of h-refined solutions of the vector Helmholtz equation using finite element 1-Form basis functions of degree 1 through 6.

5.3 Acoustic Eigenvalues

The acoustic problem corresponds to case $p = 3$ (1)-(4) with the exception that the source term Φ is zero and we are seeking the fundamental eigensolutions of the operator. Here u is a 2-form flux-like quantity and j is a 0-form potential-like quantity. In the case of acoustics the operator $\star_\alpha$ is density, in elasticity it corresponds to a combination of the Lamé constants; $\lambda + 2\mu$. The boundary condition on Γ_D is a zero-flux condition, while the condition on Γ_N is a zero-pressure condition. In standard notation we have

$$\begin{aligned} -\nabla(\nabla \cdot \vec{U}) &= \omega^2 \vec{U} \quad \text{in } \Omega, \\ \vec{U} \cdot \vec{n} &= 0 \quad \text{on } \partial\Omega_D, \\ \nabla \cdot \vec{U} &= 0 \quad \text{on } \partial\Omega_N, \end{aligned} \tag{35}$$

where $\vec{U}$ is the velocity and ω is the resonant frequency.

In this computational example we compute the eigenvalues of the above equation on a fixed mesh for various values of polynomial degree k. We choose a problem in which the eigenmodes are known to be smooth, and thus we achieve the expected exponential convergence. The computational domain is a unit cube with the exact eigenvalues given by

$$\omega^2 = \pi^2 \left(l^2 + m^2 + n^2\right), \tag{36}$$

with $l, m, n \neq 0$. The domain is discretized using a 6 element tetrahedral mesh, and equation (35) is discretized using 2-form basis functions, with the required discrete bilinear forms computed by the Bilinear2Form methods getMassMatrix() and getStiffnessMatrix(). This results in a generalized linear eigenvalue problem

$$\mathbf{A}\mathbf{x} = {\omega_h}^2 \mathbf{B}\mathbf{x}, \tag{37}$$

where $\mathbf{A}$ and $\mathbf{B}$ are the global 2-form stiffness and mass matrices, respectively. The vector $\mathbf{x}$ represents the unknown coefficients of the basis function expansion of the eigenmode $\vec{U}$, and ω_h is the computed resonant frequency of the eigenmode.

In this example we use Matlab to compute the entire set of eigenvalues of (37). The model equation (35) has an infinite set of zero-valued eigenvalues, corresponding to solenoidal solutions. The discrete spectrum therefore has a large number of zero-valued (to machine precision) eigenvalues. While this is evidence that our discretization correctly models the kernel of the grad-div operator, these eigenvalues are of no interest to us, so we search the computed spectrum for the first non-zero eigenvalue which according to (36) should have the value $3\pi^2$. In Figure 12 we plot the log of the error $|\omega - \omega_h|$ of the first non-zero eigenvalue versus k, the degree of the finite element approximation. We see the expected exponential convergence. The plateaus in the convergence are due to the symmetry of the fundamental mode. For very large problems in which it is not feasible to use Matlab, it is possible to develop iterative eigenvalue solvers that quickly converge to the smallest non-zero eigenvalues [White and Koning 2002].

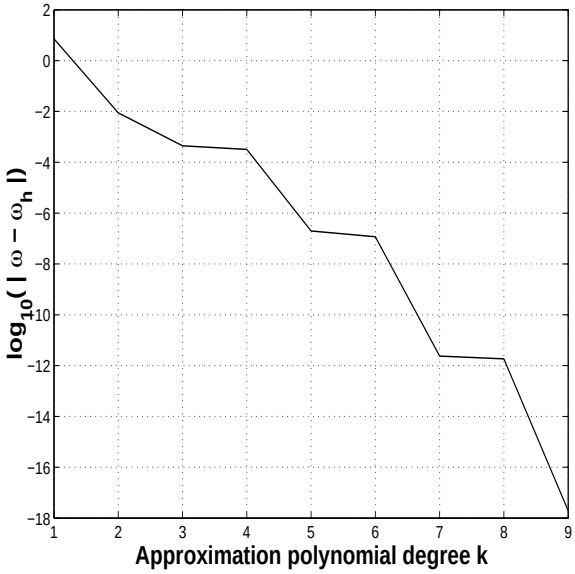

Fig. 12. Exponential p-convergence of the 2-form discretization of the acoustic equation.

6. ELECTROMAGNETICS APPLICATIONS

In this section present some practical applications of the Discrete Differential Forms approach. These applications range from electrostatics, electromagnetic eigenvalue problems, to full-wave time-dependent electromagnetic waves. As these are real application problems, we do not know the exact solution so the error in the numerical approximation is not known. We do not provide detailed information on each of these applications, rather the purpose of this section to provide a high-level overview of the types of problems that can be solved via the Discrete Differential Forms methodology.

6.1 Electrostatics

As an example electrostatics problem we consider the computation of resistance in a integrated circuit. The appropriate equation is Poisson's equation (29) with α representing the conductivity and Φ representing the electrostatic potential. Given boundary conditions on Φ, we solve for Φ in the interior of the problem and then post-process to determine the current $J = \alpha\nabla\Phi$. It is a simple matter to then integrate J to yield the total current and therefore the resistance. The potential Φ is approximated by a 0-form basis, the current density J by a 1-form basis. The source term f is zero for this problem. The complete geometry of the problem is shown in Figure 14, and a close-up view of the geometric is shown in 13. In the close-up view, the rectangular pads are 20μm by 30μm. There are several different materials present in this problem, the bulk semiconductor substrate (medium conductivity), a thin epitaxial layer (Low conductivity), and well (high conductivity). There are also numerous perfectly conducting pads on the top surface, these are where the voltage boundary conditions are applied. The computational mesh was a very simple Cartesian mesh consisting of approximately 2 million elements.

We solved this problem many times with many different boundary conditions, typically one pad is held at +1V, another pad is held at -1V, and all other pads are grounded. Alternatively, we can leave the pads "floating" rather than grounded, and this can have a significant effect on the resistance. In Figures 15 and 16 we show the computed electrostatic potential for the boundary condition +1V applied to one pad and -1V applied to 3 neighboring pads. It should be noted that CASC's algebraic multigrid solver BoomerAMG was used to solve the resulting system of equations.

6.2 Eigenvalues

As an example electromagnetic eigenvalue problem we compute the resonant frequencies of an accelerator induction cell. Induction cells are a key component of linear accelerators, they are used to accelerate the electron beam by providing a large (e.g. 1MV) potential across a small gap. The passing electron beam can, however, induce unwanted resonant modes in the cell. Designers can choose the shape and the material loading of the induction cell to minimize the interaction between the unwanted resonant modes and the beam. In Figure 17 we show the exterior of the computational mesh used for the simulation, and in 18 we show a close-up cut-away view of the accelerating gap. The interior consists of both vacuum and dielectric regions. Note the curvature of the walls of this device, it would

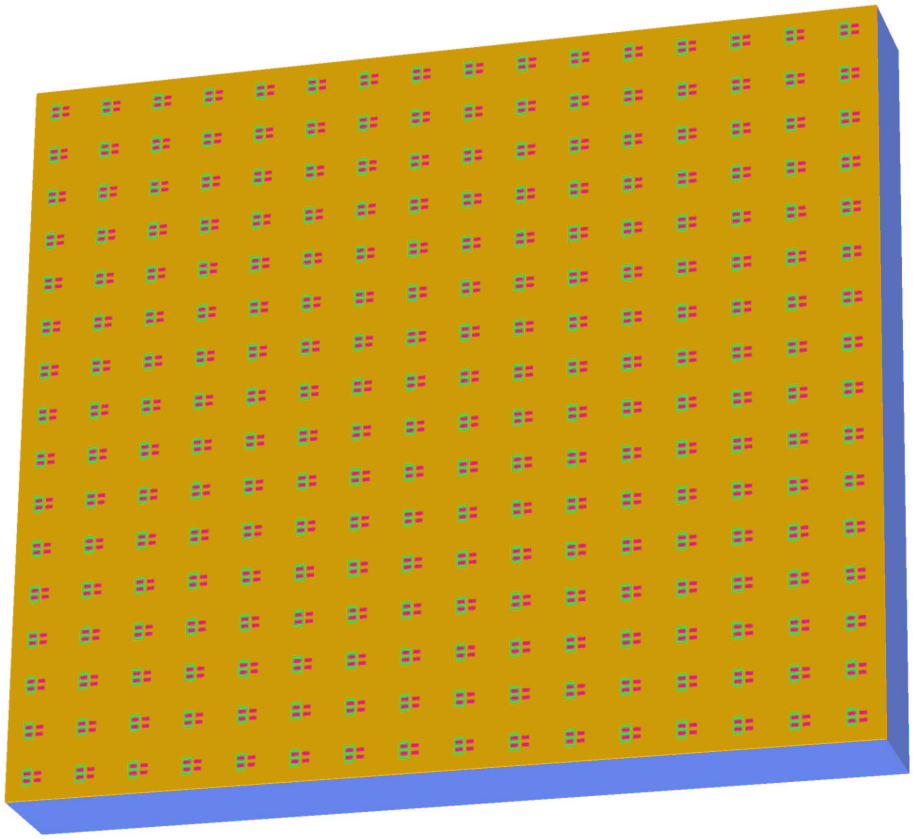

Fig. 13. Geometry for the semiconductor resistance test problem showing the arrangement of conducting pads

be difficult to obtain accurate computed fields using an FDTD-type method.

The PDE for this problem is given by (32) with zero source function f, we are interested in computing the resonant frequencies ω and the corresponding eigenmodes. We discretize (32) using 1-form basis functions, resulting in a generalized eigenvalue problem with exactly the same form as the acoustic eigenvalue problem defined in Section 5.3. However, unlike the results presented in Section 0**??** we do not use Matlab to compute the entire spectrum. Instead we use a novel iterative procedure designed especially for this type of electromagnetic eigenvalue problem [White and Koning 2002]. This procedure efficiently computes the 20 smallest non-zero eigenvalues and corresponding eigenvectors.

In Figures 19 - 22 we show the computed eigenmodes corresponding to the 1st, 5th, 12th, and 20th smallest eigenfrequencies. It is the magnitude of the induced

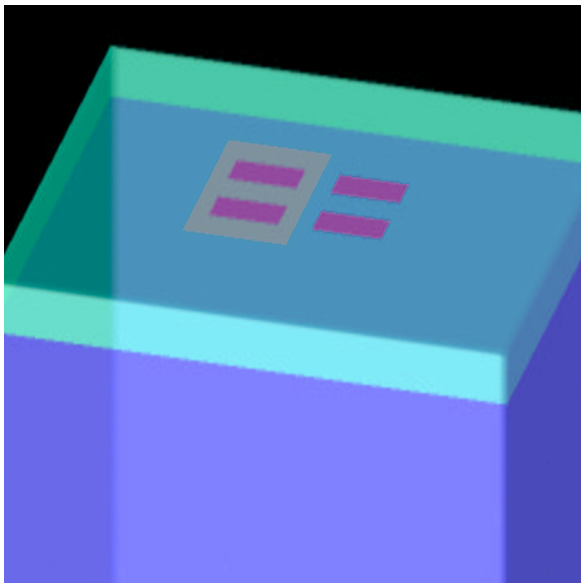

Fig. 14. Close-up view of materials present in the semiconductor resistance problem showing the substrate, the epitaxial layer, the well, and the conducting pads.

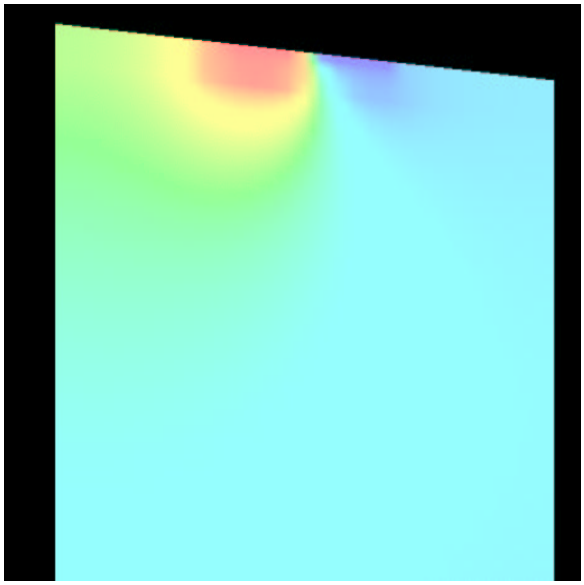

Fig. 15. Electrostatic potential versus depth in the semiconductor substrate, for a particular slice through the mesh. The color denotes the value of the electrostatic potential, ranging from red (+1) to blue (-1).

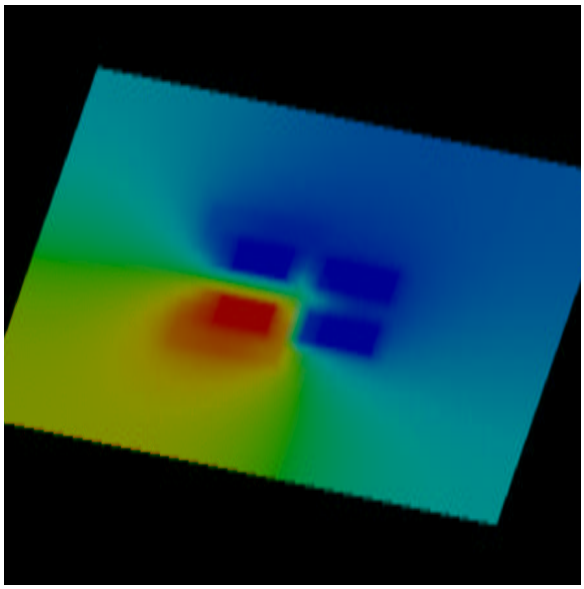

Fig. 16. Electrostatic potential in the x-y plane in the epitaxial layer below the conducting pads. The color denotes the value of the electrostatic potential, ranging from red (+1) to blue (-1).

surface current density that is displayed. It is interesting to note than some modes such as the 1st and the 20th have maximum field intensity in the accelerator gap and hence will interact strongly with the electron beam, while other mode such as the 5th and the 12th have zero field in the gap and will not interact with the beam.

6.3 Maxwell's Equations

The main use of the FEMSTER framework is time-domain computational electromagnetics. Typically the second order wave equation for the electric field is solved:

$$\epsilon \frac{\partial^2}{\partial t^2}\vec{E} \;=\; -\nabla \times \mu^{-1}\nabla \times \vec{E} - \frac{\partial}{\partial t}\vec{J} - \sigma \frac{\partial}{\partial t}\vec{E} \;\text{ in } \Omega, \tag{38}$$

where ϵ and μ are the tensor electric permittivity and magnetic permeability respectively, and σ is the electrical conductivity. The boundary conditions on $\vec{E}$ are typically a combination of perfect conducting, impedance, or radiation boundary conditions. The discretized wave equation is

$$\mathbf{B}\ddot{\mathbf{x}} = -\mathbf{A}\mathbf{x} - \mathbf{C}\dot{\mathbf{x}} - \dot{\mathbf{y}}, \tag{39}$$

where $\mathbf{A}$ is the stiffness matrix and $\mathbf{B}$ and $\mathbf{C}$ are mass matrices involving the permittivity and conductivity, respectively. These matrices are similar to those used in the frequency domain Helmholtz equation in Section 5.2. In (39), time is discretized using the second-order accurate leap-frog method, the stability and conservation properties are discussed in [Rodrigue and White 2001]. Naturally, it is possible to use other more accurate time stepping methods such as 4th order Runge-Kutta, this is an area of future research. Note that in general the time stepping

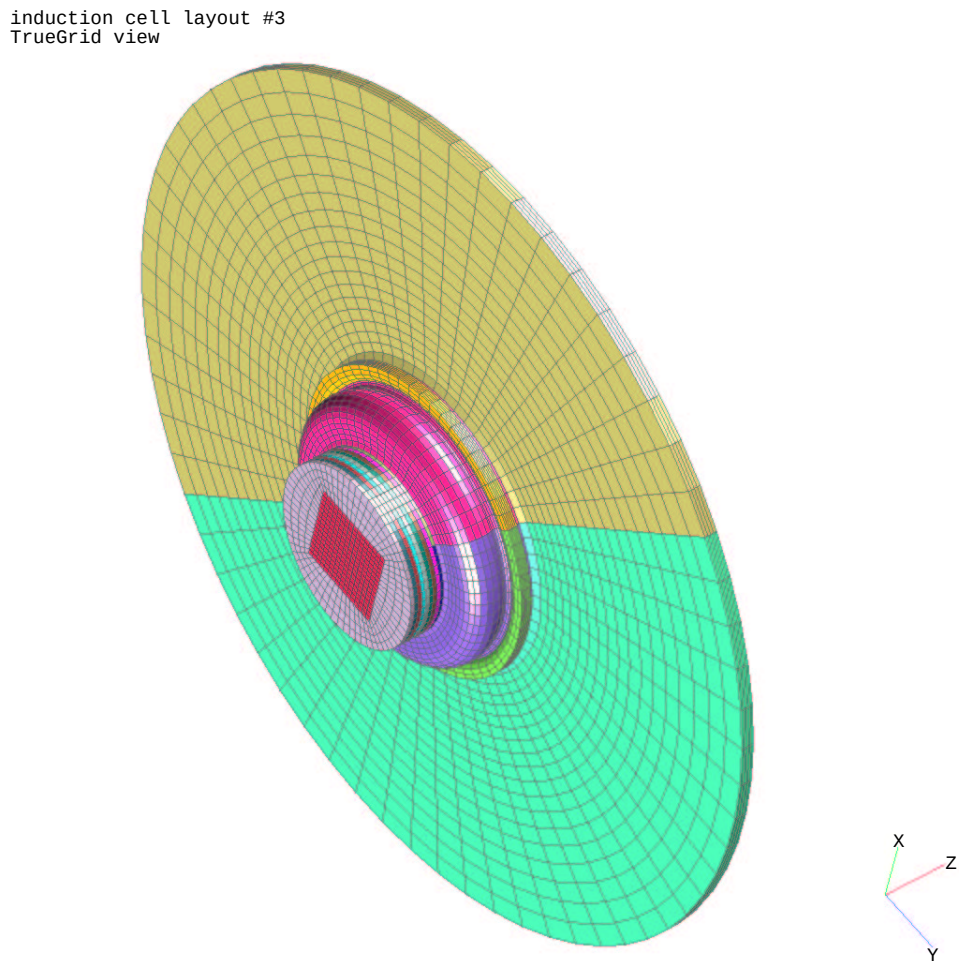

Fig. 17. Exterior of the computational mesh used for the resonant mode calculations. The colors are to aid in visualization only, they do not represent anything physical.

algorithm will require that a linear system of equations be solved at every time step. It is well known that for linear basis 1-form basis functions on a Cartesian grid the trapezoidal rule can be used to evaluate the bilinear forms, yielding a diagonal mass matrix $\mathbf{B}$. This is often referred to as "mass lumping". This seemingly crude approximation yields no degradation of accuracy, in fact the discrete equations are identical to the popular Finite Difference Time Domain method. However, for non-Cartesian grids mass lumping cannot be applied without severe degradation, in fact the consistency of this approach is debatable. Fortunately, the mass matrix $\mathbf{B}$ is very well conditioned even for distorted grids, the matrix can be solved in $O(n)$ operations using standard conjugate gradient algorithms [Koning et al. 2000].

6.3.1 *Microwave Devices.* Our first example of a transient simulation is of a three conductor coplanar waveguide which consists of a 2 micrometer thick metal

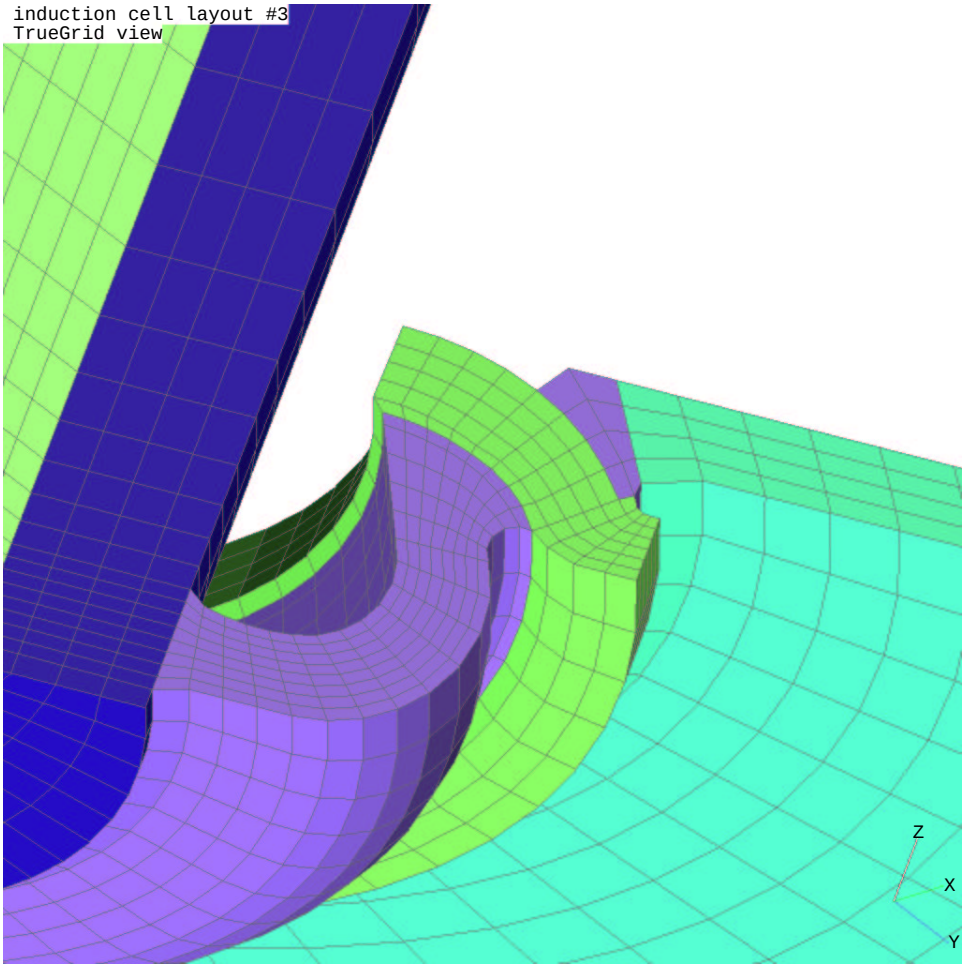

Fig. 18. Interior of the computational mesh used for the resonant mode calculations.

deposited on silicon. The outer conductors can be considered ground, and the inner conductor is the signal conductor. For high frequency signals, the coplanar waveguide configuration is superior to the traditional microstrip, as the fields are confined to the small region between the conductors. The domain is discretized using hexahedral element mesh. Figure 23 shows the x-y plane containing the metal.

We excite the problem with a time-varying current source at the left end of the waveguide. We are interested in how the induced voltage pulse travels down the guide. We run the simulation for 6000 time steps. The induced currents on the conductors can be measured, and this data can be post-processed to yield input impedance, S-parameters, and other useful characteristics of the waveguide. The voltage input and output for the transmission line is shown in Figure 24. In Figure 25 we show snapshots of the computed magnetic field at diffeeent instants of time.

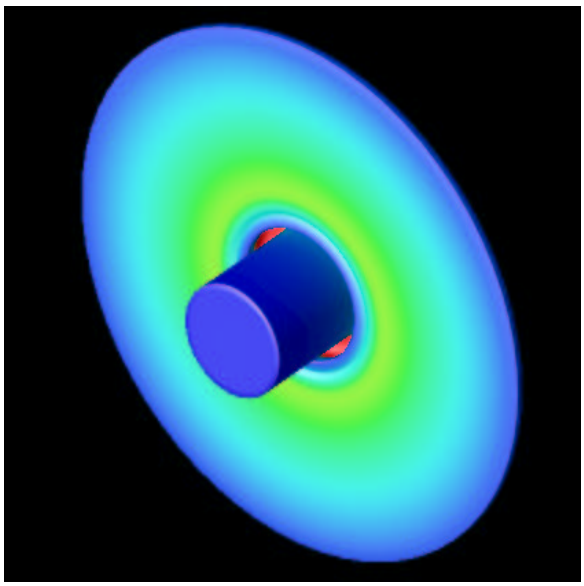

Fig. 19. The 1st eigenmode, 85.6 MHz.

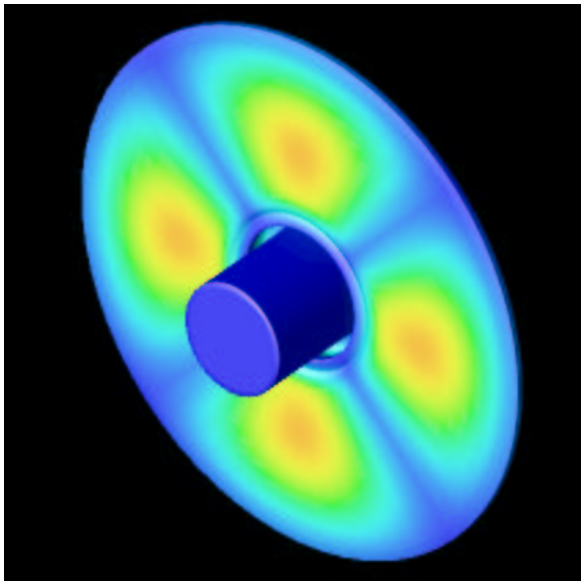

Fig. 20. The 5th eigenmode, 181.9 MHz.

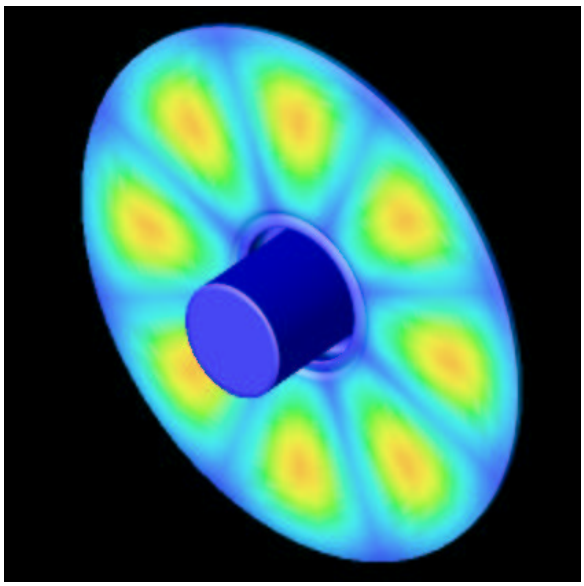

Fig. 21. The 12th eigenmode, 271.9 MHz

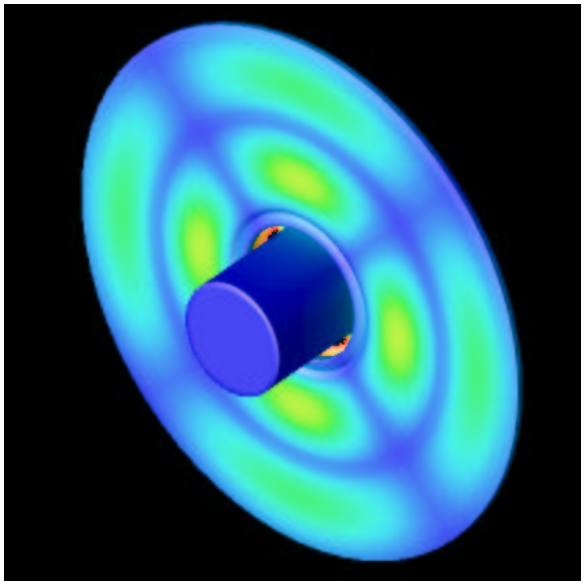

Fig. 22. The 20th eigenmode, 349.5 MHz.

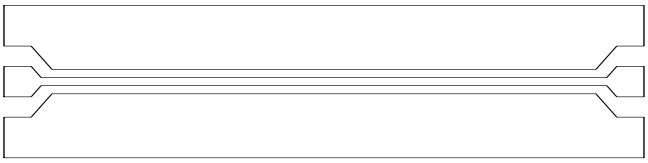

Fig. 23. Coplanar waveguide geometry (not to scale).

This simulation was a full-wave three-dimension simulation, in this figure we are examing the magnetic field in the mid-plane of the microstrip.

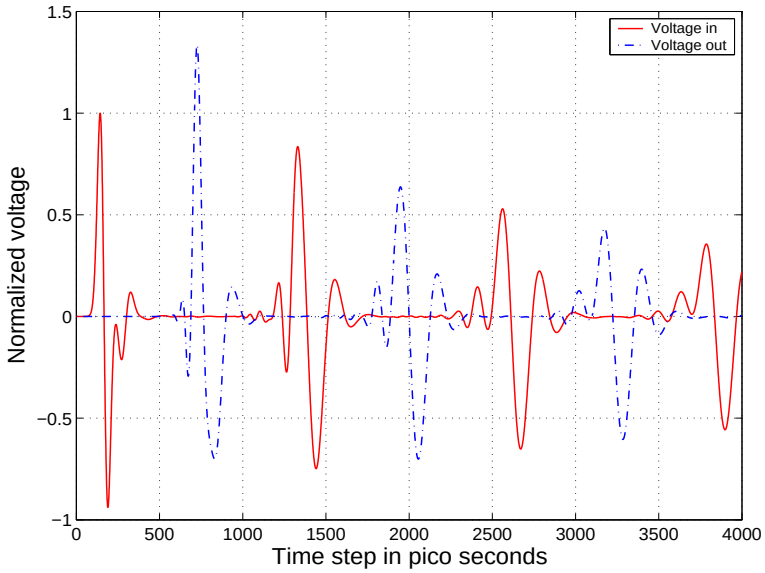

Fig. 24. Input and output voltage for the coplanar waveguide. The dashed lines is the voltage at the input port, the solid line is the voltage at the output port. Note the reflections due to the imperfectly matched loads.

The second example solution of Maxwell's equations is of a planar microstrip inductor. Planar microstrip inductors are used in microwave filters, amplifiers, and oscillators. In the standard circuit design process the DC values of inductance are used in the circuit design, however the validity of this approximation decreases with increasing operational frequency. While the FEMSTER library could be used to compute the DC inductance, here we compute the transient response to the inductor due to a very broadband excitation. This particular inductor consists of 2 micrometer thick metal deposited on silicon in a square spiral configuration. The overall width of the inductor is the same as the coplanar waveguide illustrated in Figure 23, in fact these two devices are designed to be compatible. For analysis purposes one end of the inductor is short-circuited, at the other end we apply a time varying current source. The current source has the form of the integral of a

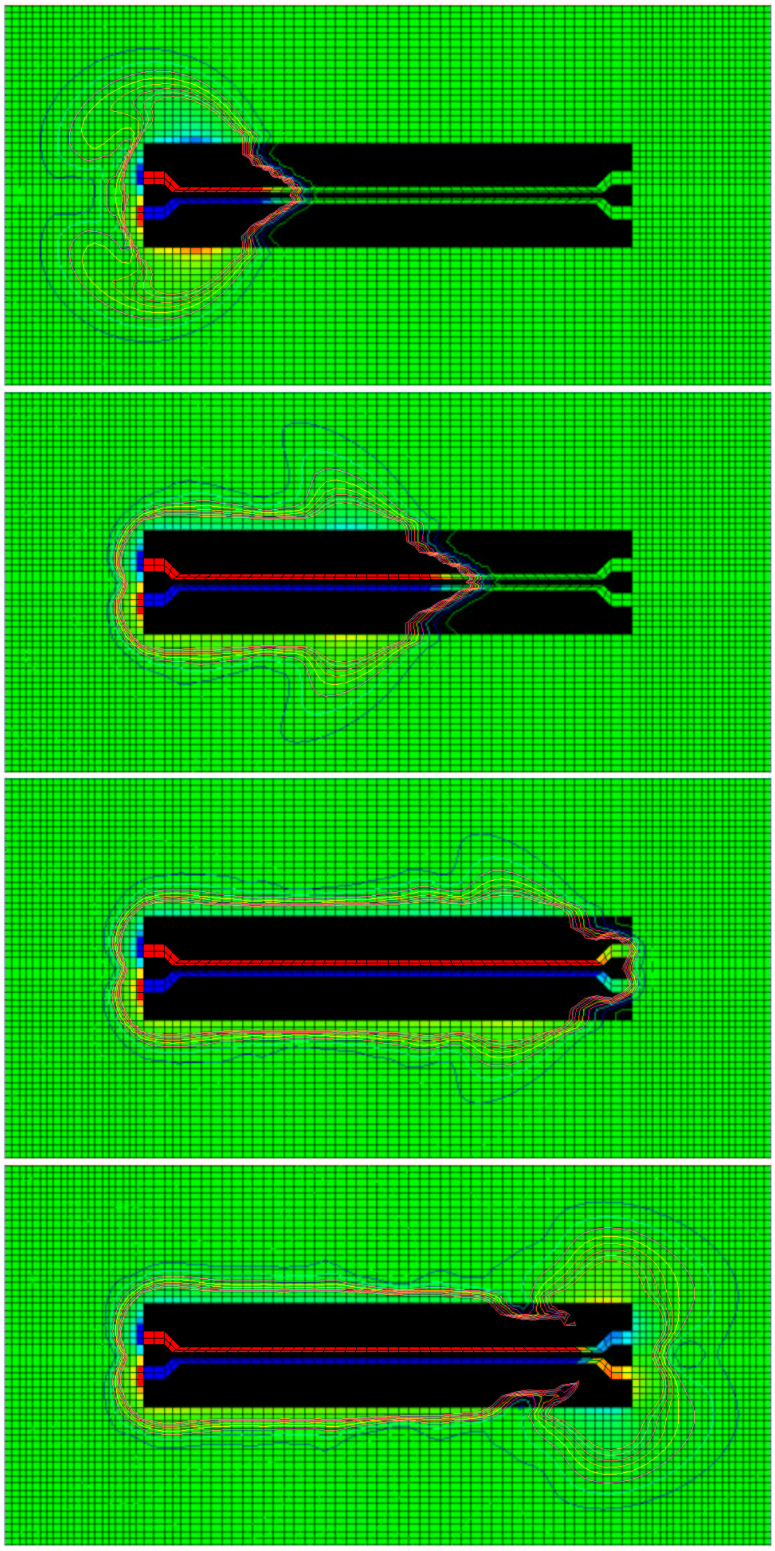

Fig. 25. Snapshots of the computed magnetic field for the coplanar microstrip waveguide. The psuedocolor represents the z-component of the magnetic field vector. The contours represent the magnetic field energy density. Note that while most of the field is confined to the space between the conductors, the there is some radiation particularily at the right end of the waveguide.

Gaussian, it starts with value zero at time zero and smoothly grows to have value 1. While in practice the design engineer may be interested in the induced voltage vs. time at the input port, here we run the simulation for 8000 time steps and present only the final steady-state field configuration. Figures 26 - 28 show the magnitude of the z-component of the magnetic field in the x, y, and z planes, respectively. We note that, as expected, the field is largest in the center of the inductor. However the fields are non-zero outside of the inductor and there will be significant coupling to adjacent circuit elements.

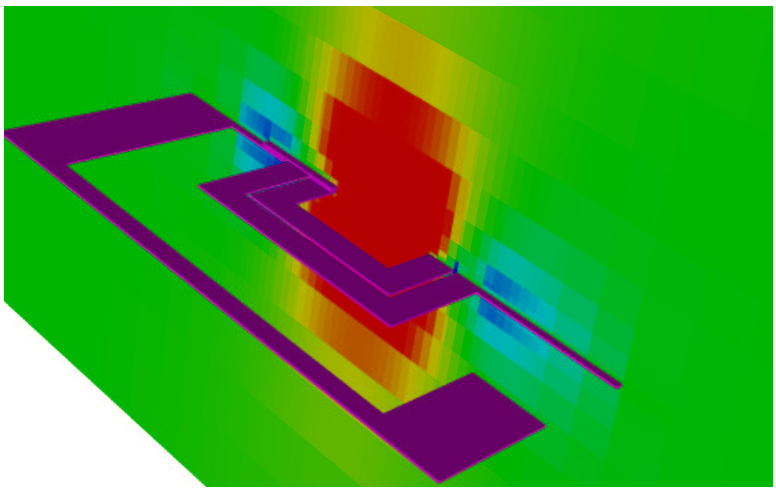

Fig. 26. Magnetic field z-component magnitude in x-plane.

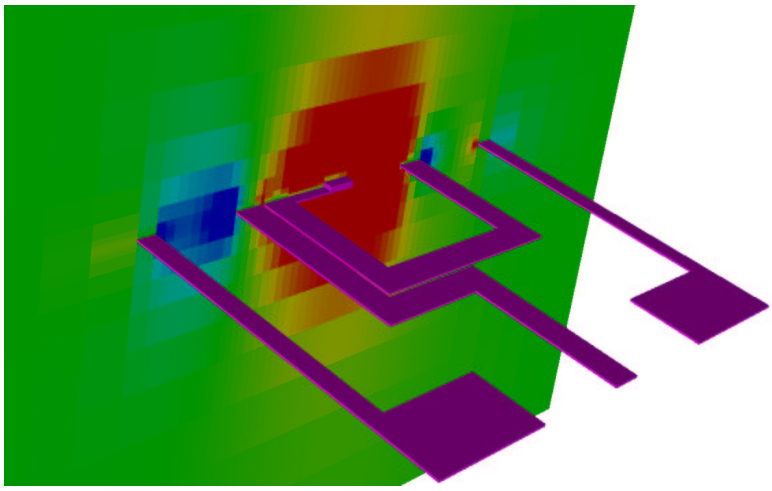

Fig. 27. Magnetic field z-component magnitude in y-plane.

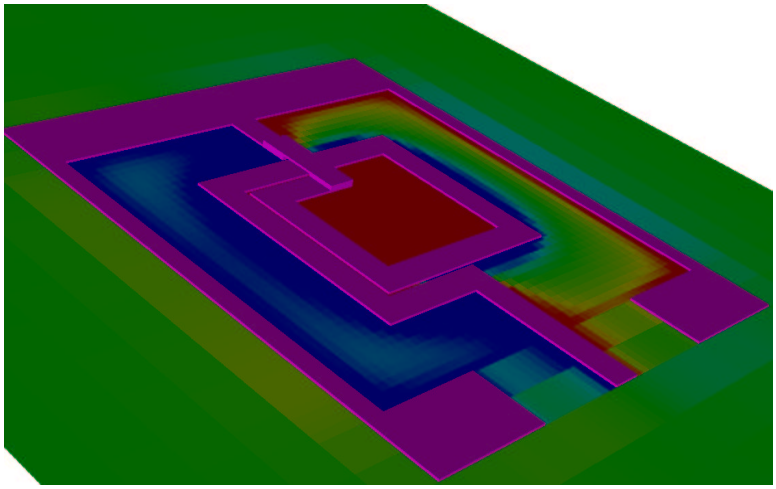

Fig. 28. Magnetic field z-component magnitude in x-plane.

6.3.2 *Optical Devices.* For the first simulation of an optical device, we computed electromagnetic fields in a bent single mode optical fiber. A 3-dimensional hexahedral mesh is used for the simulation, with approximately 3 million elements. Due to the very fine mesh used to resolve the fiber core, 1st order basis functions are used. A TE01 wave is launched at one end of the 5 micrometer radius fiber core, and the other end is terminated with a Maxwellian perfectly matched layer. The fiber is designed to support a 1.55 micrometer wavelength mode and is 10 wavelengths long with a bend radius of 19.73 micrometers. The purpose of the simulation is to determine how much, if any, of the electric field escapes the fiber due to the sharp bend. The simulation is run for 3000 time-steps with $dt = 3.3e^{-16}s$. This is a full-wave, explicit time-domain simulation. Snapshots of the fiber core electric field intensity are shown in Figures 29 - 31.

As the second example of an optical device, we compute the fields in a so-called "holey" fiber. These optical fibers have a very complex structure permitting broadband single-mode operation. In addition the fiber is larger than traditional single-mode fibers, enabling greater power handling which is important for fiber amplifier applications. For more information on holey fibers see [Broeng et al. 1999]. The computational mesh for the holey fiber problem is illustrated in Figure 32. The cylindrical rods are glass, the surrounding medium is of a very low dielectric. The cylinders are 1.92μm radius separated by 3.2μm. The mesh was approximately 10 wavelengths long in the propagating direction, and terminated with a PML. Ideally, the mesh would be significantly longer in order to better approximate an infinite fiber.

The next example is of a photonic bandgap waveguide with a 90-degree right angle bend. The photonic bandgap region consist of a periodic array of rods in a background medium. The dielectric constant of the rods, the diameter of the rods, and the rod spacing are chosen such that waves of a particular frequency are prohibited from propagating. Hence the term bandgap. The waveguide is formed

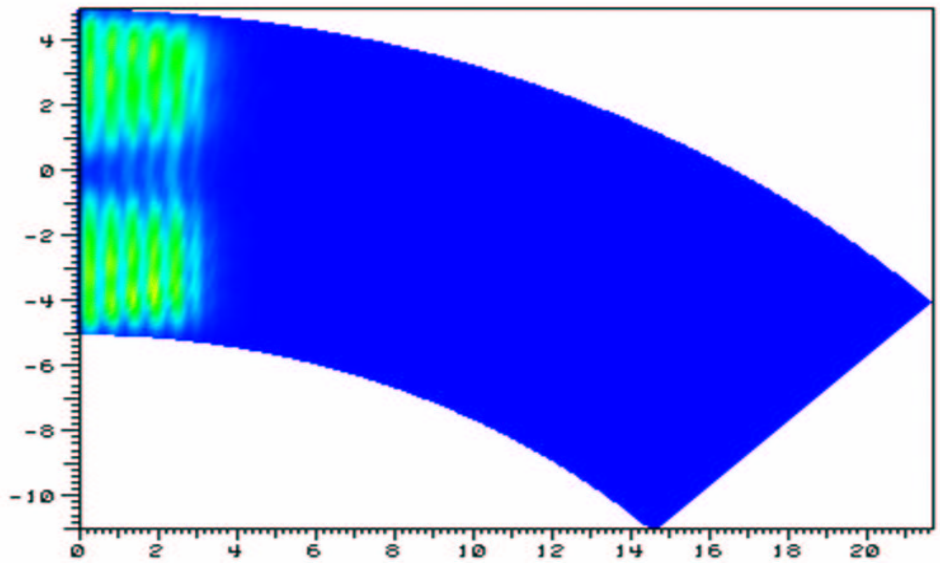

Fig. 29. Bent optical fiber electric field data after 10 time steps.

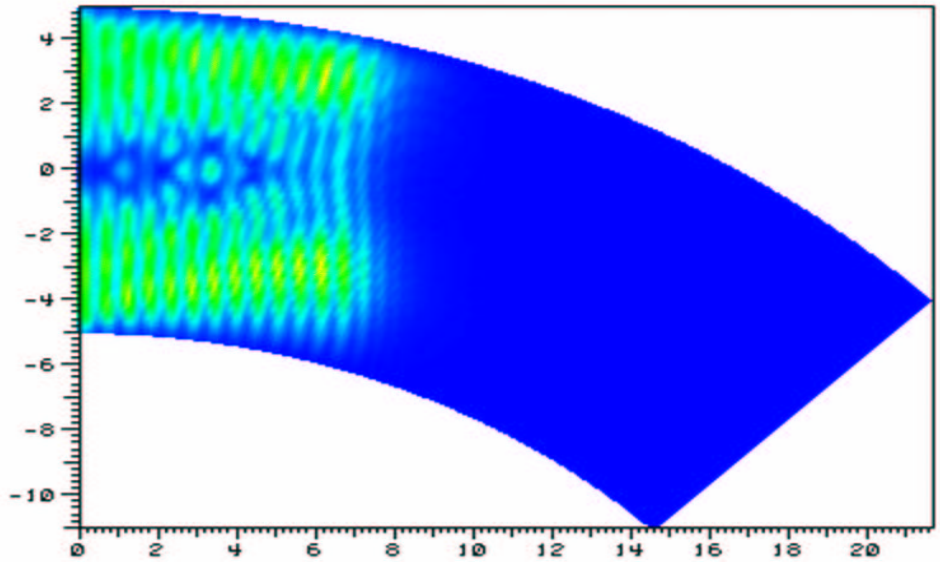

Fig. 30. Bent optical fiber electric field data after 24 time steps.

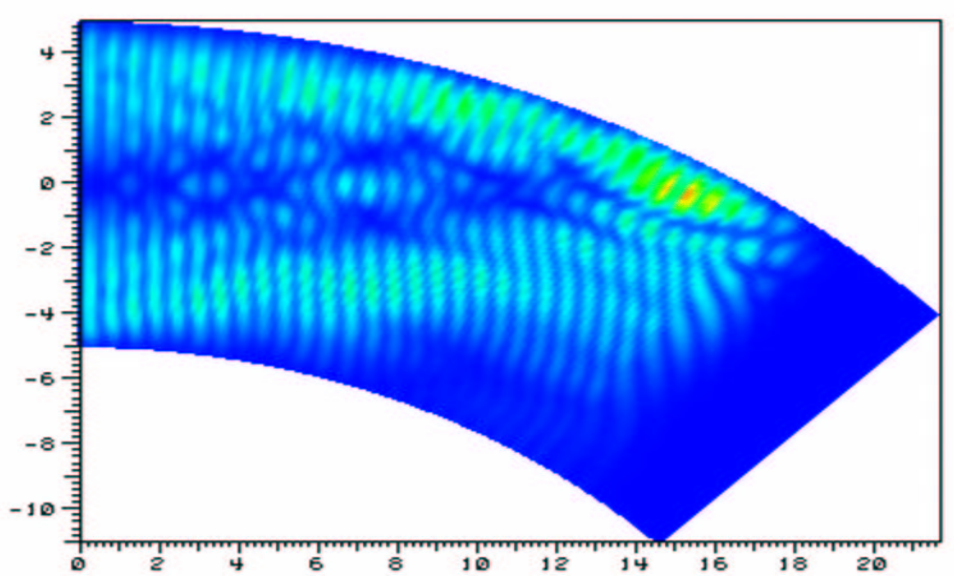

Fig. 31. Bent optical fiber electric field data after 57 time steps.

by removing a line of rods. A good exposition on photonic bandgap structures is given in [Joannopoulus et al. 1995]. In this particular application the rods have a 0.1μm radius and are 0.6μm apart. Our computational mesh is shown in Figure 36. This a conforming 3D hexahedral mesh that is one element thick in the z-direction, simulating a 2D problem. The different colors on the boundary of the problem represent fictitious materials used in the PM.

The simulation begins with a plane wave entering the waveguide from the left. The wave propagates in the waveguide making a nice 90-degree turn. This is not possible with standard optical waveguides, as there would be significant scattering from the 90-degree bend and only a fraction of the energy would actually come out of the far end of the waveguide. In Figure 37 - 40 we show snapshots of the electric field intensity. Note that while the fields are non-zero outside of the waveguide, these are evanescent fields and the device is behaving as a waveguide.

6.3.3 *Accelerator Wakefield's.* As a final example, we compute the wake-fields in a generic linear accelerator. The accelerator consists of a series of induction cells approximately one meter in diameter. An electron bunch travels down the center of the accelerator, receiving a 1 MEV boost as it passes by each induction cell. The traveling electron bunch generates electromagnetic waves which may resonate within the induction cell for a very long time. These waves are referred to as the wake-field. This wake-field can interfere with the next electron bunch causing a beam instability. In this simulation we model the electron bunch as a rigid Gaussian beam, i.e. we are simply interested in the wake-field and not in the precise motion of the electrons. The accelerator is modeled by a 3-dimensional hexahedral mesh with approximately 500,000 elements, with perfectly conducting walls and a Maxwellian perfectly matched layer at each end. In this simulation we

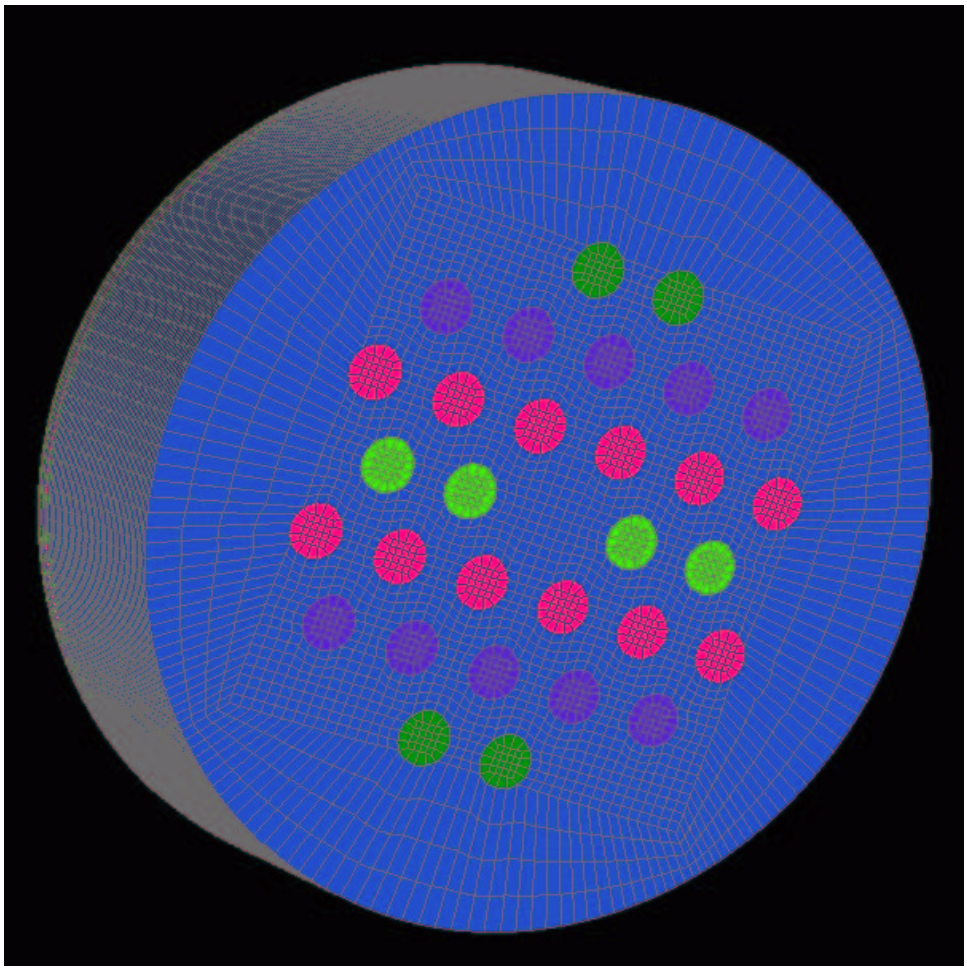

Fig. 32. Computational mesh for the holey fiber simulation..

solve for both the electric and magnetic fields in a leapfrog manner similar to the FDTD method, except that we solve a finite element mass matrix at every time step. We do this simply because the magnetic field is the more intuitive field to visualize for this problem. The surface magnetic field is illustrated in Figures 41 - 44.

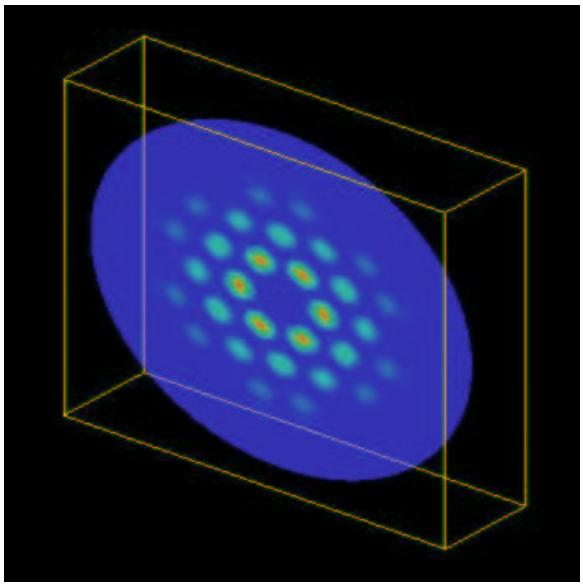

Fig. 33. Electric field intensity in the holey fiber at slice z=29.

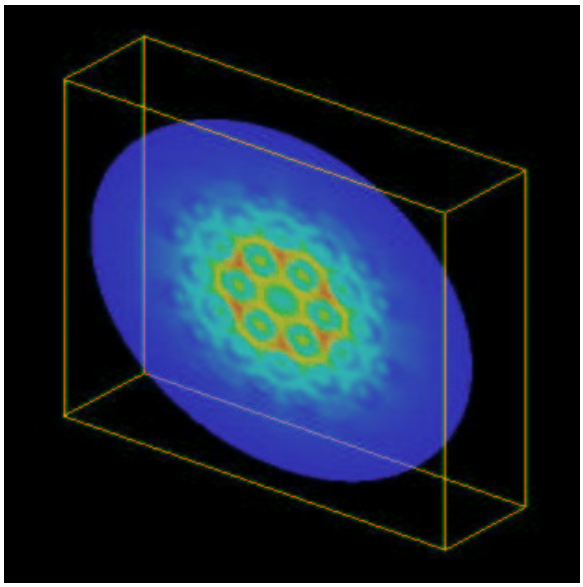

Fig. 34. Electric field intensity in the holey fiber at slice z=73.

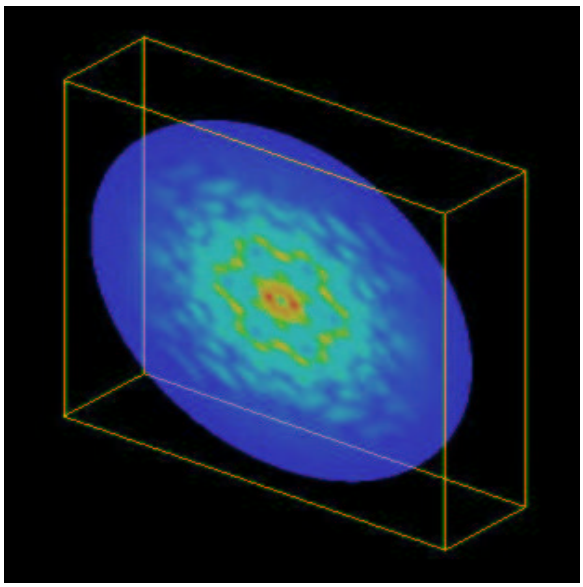

Fig. 35. Electric field intensity in the holey fiber at slice z=99. At this position the electric fields have nearly converged to the steady-state solution. Note that most of the electric field is contained in the space between the glass rods. While the field is non-zero beyond the rods, these fields are evanescent in nature similar to the fields in the cladding of a standard single-mode optical fiber.

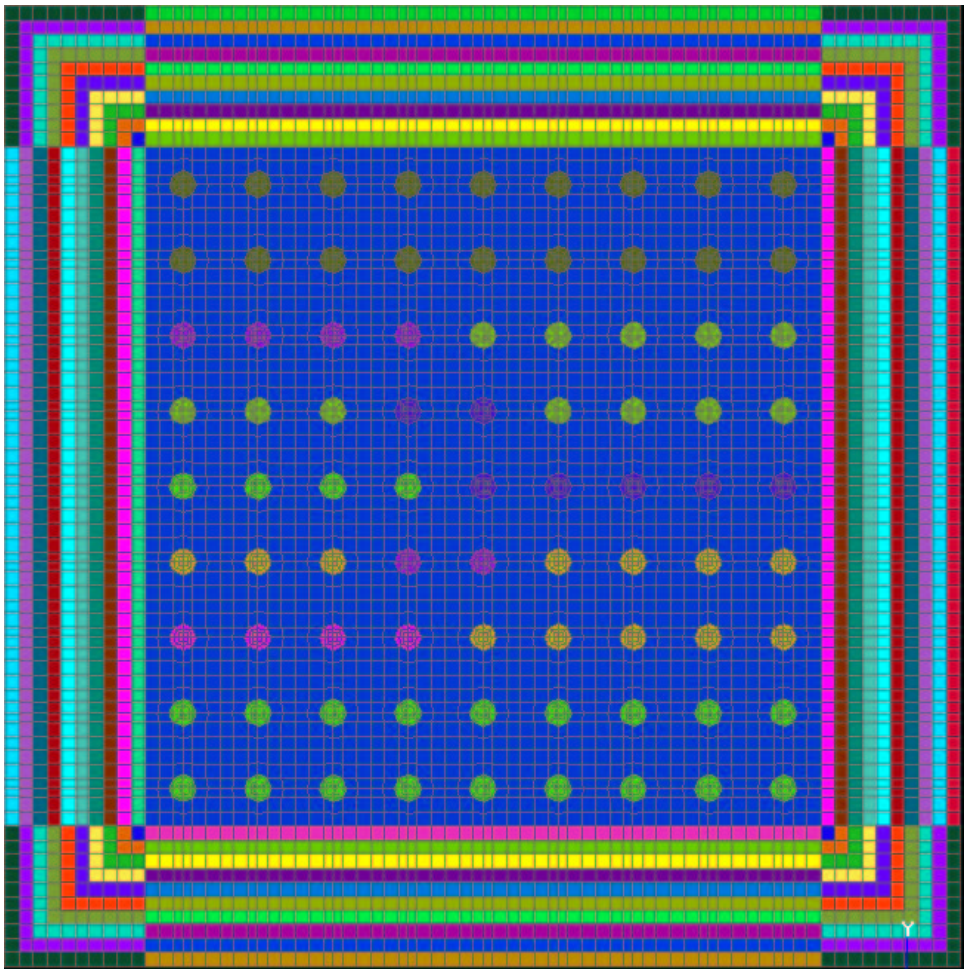

Fig. 36. Computational mesh for the photonic bandgap waveguide simulation.

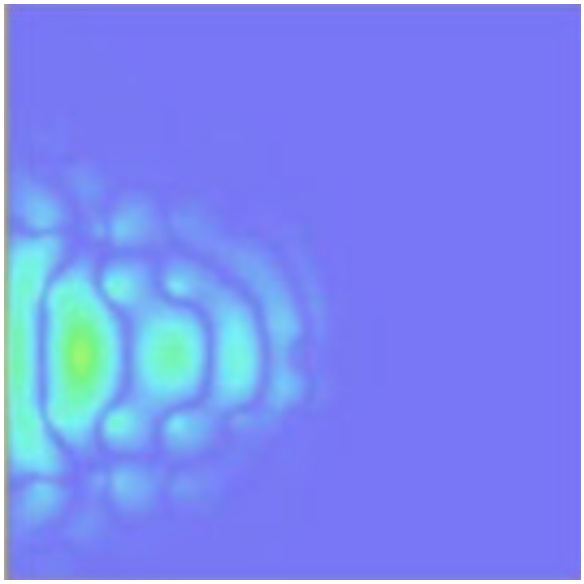

Fig. 37. Electric field intensity in the photonic bandgap waveguide at t=19.

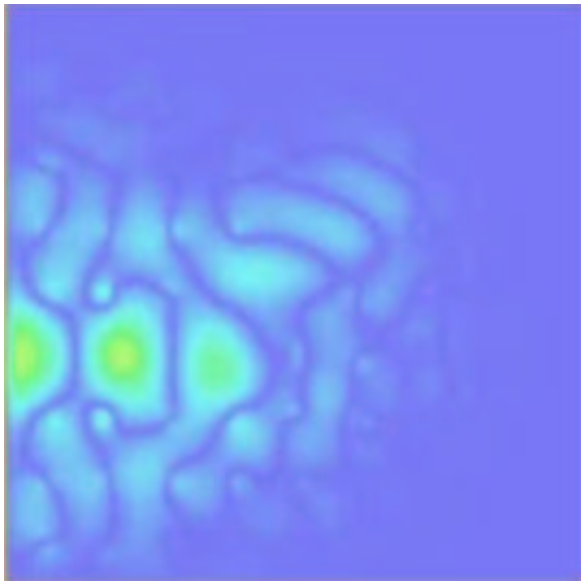

Fig. 38. Electric field intensity in the photonic bandgap waveguide at t=36.

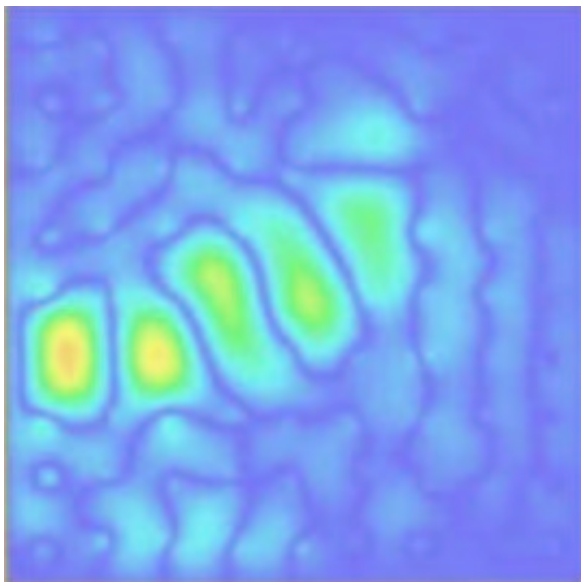

Fig. 39. Electric field intensity in the photonic bandgap waveguide at t=76.

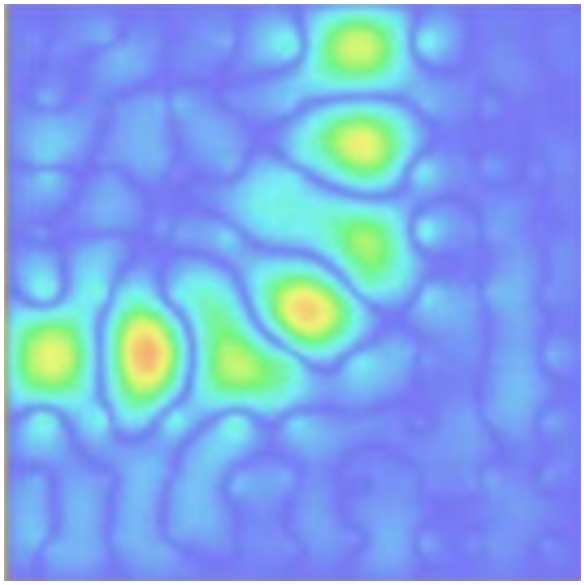

Fig. 40. Electric field intensity in the photonic bandgap waveguide t=96.

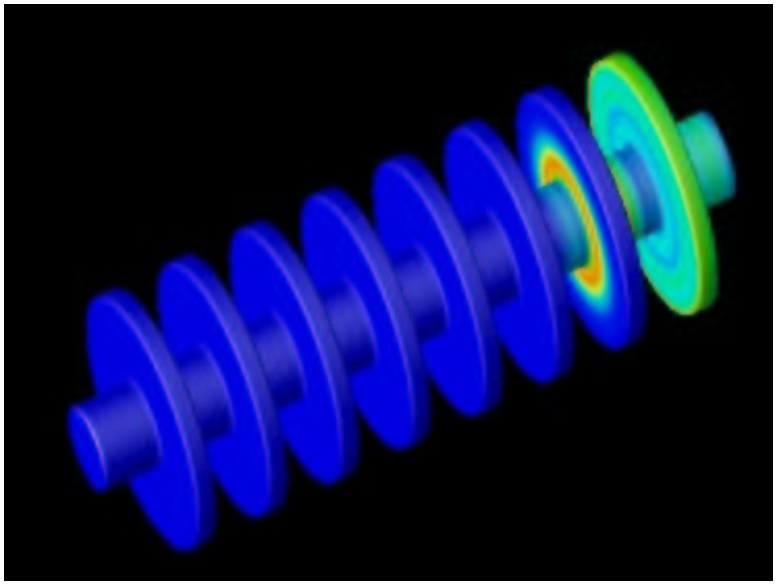

Fig. 41. Snapshot of the electric current induced on the walls of a linear accelerator due to a Gaussian electron bunch. At this instant the bunch, traveling from right to left, is approximately 25 percent of the way down the accelerator.

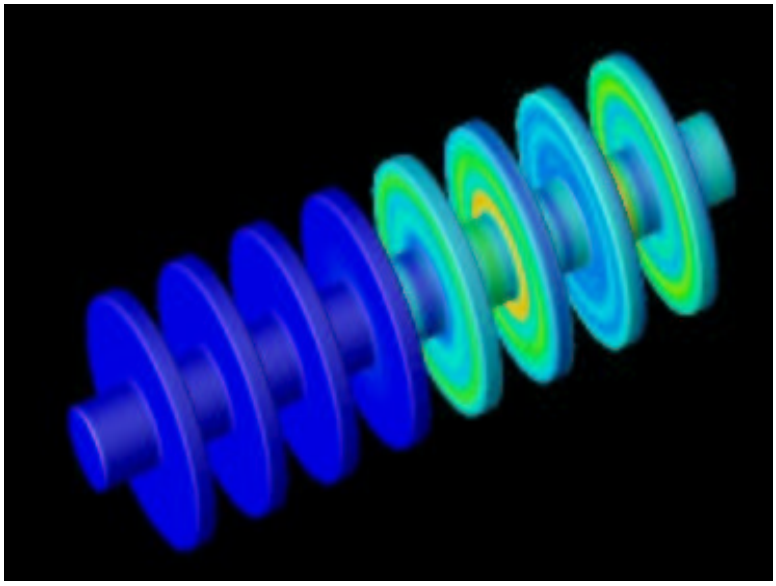

Fig. 42. Snapshot of the electric current induced on the walls of a linear accelerator due to a Gaussian electron bunch. At this instant the bunch, traveling from right to left, is approximately 50 percent of the way down the accelerator.

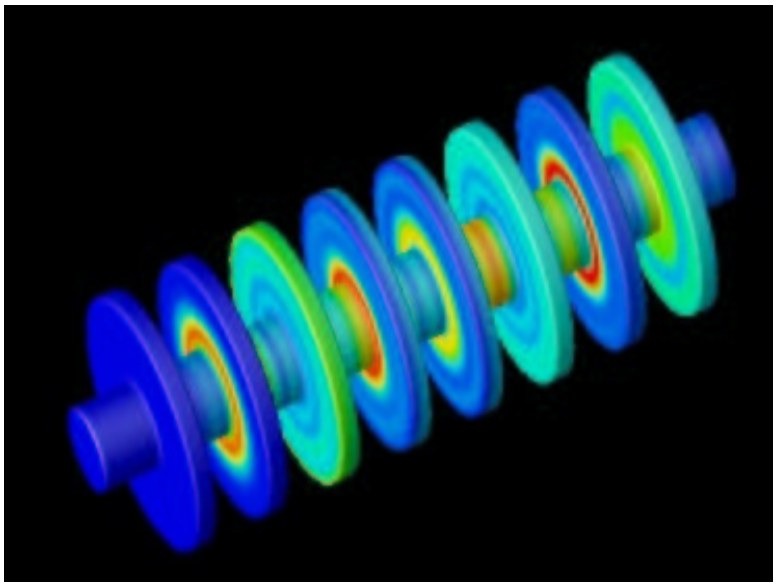

Fig. 43. Snapshot of the electric current induced on the walls of a linear accelerator due to a Gaussian electron bunch. At this instant the bunch, traveling from right to left, is approximately 90 percent of the way down the accelerator..

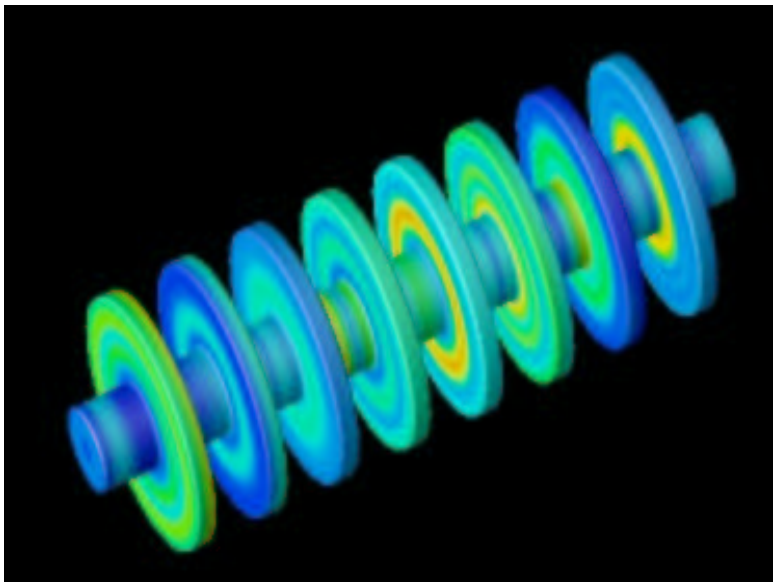

Fig. 44. Snapshot of the electric current induced on the walls of a linear accelerator due to a Gaussian electron bunch. At this instant the bunch, traveling from right to left, has exited the accelerator. Note that the induction cells resonate for a long time after the electron bunch has passed by.

7. CONCLUDING REMARKS

The FEMSTER finite element class library described in this paper is unique in several respects. First,it is based upon the language of differential forms. This language provides a unified description of a great variety of PDE's, and thus leads us directly to a concise and abstract interface to our finite element methods. This language also unifies the seemingly disparate Lagrange, $H(curl)$ and $H(div)$ basis functions that are used in computational electromagnetics. Secondly, FEMSTER utilizes higher-order elements, bases, and integration rules. Higher-order elements are important for accurate modeling of curved surfaces. The use of higher-order basis functions reduces the demands put upon mesh generation, e.g. a billion element mesh is no longer required for a numerically converged solution. The FEMSTER class library is ideally suited for researchers who wish to experiment with unstructured-grid, higher-order solution of Poisson's equation, the Helmholtz equation, Maxwell's equations, and related PDE's that employ the standard gradient, curl, and divergence operators.

REFERENCES

Abraham, R., Marsden, J. E., and Ratiu, T. 1996. *Manifolds, Tensor Analysis, and Applications*, Second edition ed. Applied Mathematical Sciences. Springer Verlag.

Babuska, I., Ihlenburg, F., Strouboulis, T., and Gangaraj, S. K. 1997. A posteriori error estimation for finite element solution of Helmholtz. *Internat. J. Numer. Methods Engrg. 40,* 21, 3883–3900.

Babuska, I., Strouboulis, T., Upadhyay, C., and Gangaraj, S. K. 1995. A posteriori estimation and adaptive control of the pollution error in the h-version of the finite element method for finite element solution of Helmholtz. *Internat. J. Numer. Methods Engrg. 38,* 24, 4207–4235.

Baldomir, D. 1986. Differential forms and electromagnetism in 3-dimensional Euclidean space R3. *IEEE Proceedings. 133,* 3, 139–143.

Bossavit, A. 1998. *Computational Electromagnetism : variational formulation, complementarity, edge elements.* Academic Press.

Brezzi, F. and Fortin, M. 1991. *Mixed and Hybrid Finite Element Methods.* Springer Series in Computational Mathematics. Springer Verlag.

Broeng, J., Mogilevstev, D., Barkou, S. E., and Bjarklev, A. 1999. Photonic crystal fibers: A new class of optical waveguides. *Optical Fiber Technology 5*, 305–330.

Burke, W. 1985. *Applied Differential Geometry : variational formulation.* Cambridge University Press.

Ciarlet, P. G. 1978. *The Finite Element Method for Elliptic Problems.* North-Holland.

Deschamps, G. 1981. Electromagnetics and differential forms. *IEEE Proceedings. 69,* 6, 676–687.

Graglia, R., Wilton, P., and Peterson, A. 1997. Higher order interpolatory vector bases for computational electromagnetics. *IEEE Trans. Ant. Prop. 45,* 3, 329–342.

Graglia, R., Wilton, P., Peterson, A., and Gheorma, I.-L. 1998. Higher order interpolatory vector bases on prism elements. *IEEE Trans. Ant. Prop. 46,* 3, 442–450.

Hiptmair, R. 1999. Canonical construction of finite elements. *Math. Comp. 68,* 228, 1325–1346.

Hiptmair, R. 2001. Discrete Hodge operators: An algebraic perspective. *J. Electromagnteic Waves Appl. 15,* 3, 343–344.

Joannopoulus, J. D., Meade, R. D., and Winn, J. N. 1995. *Photonic Crystals : Molding the Flow of Light.* Princeton, NJ.

Koning, J. M., Rodrigue, G. H., and White, D. A. 2000. Scalable preconditioned conjugate gradient inversion of vector finite element mass matrices. *Journal of Comp. and Appl. Math. 123*, 307–321.

Nédélec, J. C. 1980. Mixed Finite Elements in r3. *Numer. Math. 35*, 315–341.

Nédélec, J. C. 1986. A New Family of Mixed Finite Elements in r3. *Numer. Math. 50*, 57–81.

Raviart, P. and Thomas, J. 1977. A Mixed Finite Element Method for 2^{nd} Order Elliptic Problems. In *Mathematical Aspects of the Finite Element Method*, I. Galligani and E. Mayera, Eds. Lect. Notes. on Mathematics, vol. 606. Springer Verlag, 293–315.

Rodrigue, G. and White, D. 2001. A vector finite element time-domain method for solving Maxwell's equations on unstructured hexahedral grids. *SIAM J. Sci. Comp. 23,* 3, 683–706.

Stroustrup, B. 1991. *C++ Programming Language.* Addison-Wesley, Reading, MA.

Warren, S. and Scott, W. 1994. An investigation of numerical dispersion in the vector finite element method using quadrilateral elements. *IEEE Trans. Ant. Prop. 42,* 11, 1502–1508.

Warren, S. and Scott, W. 1995. Numerical dispersion in the finite element method using triangular edge elements. *Opt. Tech. Lett. 9,* 6, 315–319.

White, D. 2000. Numerical dispersion of a vector finite element method on skewed hexahedral grids. *Commun. Numer. Meth. Engng. 16*, 47–55.

White, D. and Koning, J. 2002. A novel approach for computing solenoidal eigenmodes of the vector Helmholtz equation. *IEEE Trans. Mag. 38,* 5, 3420–3425.

...