Efficient Finite Element Assembly of High Order Whitney Forms

Nicolas Marsic and Christophe Geuzaine

Department of Electrical Engineering and Computer Science, University of Liège

Abstract

This paper presents an efficient method for the finite element assembly of high order Whitney elements. We start by reviewing the classical assembly technique and by highlighting the most time consuming part. Then, we show how this classical approach can be reformulated into a computationally efficient matrix-matrix product. We also address the global orientation problem of the vector valued basis functions. We conclude by presenting numerical results for a three dimensional wave propagation problem.

1 Introduction

There is a growing consensus in the finite element community that state of the art low order finite element technology requires, and will continue to require, too extensive computational resources to provide the necessary resolution for complex simulations, even at the rate of computational power increase [1]. The requirement for precise resolution naturally leads us to consider methods with a higher order of grid convergence than the classical second order provided by most industrial grade codes. In particular, for high frequency time harmonic electromagnetic simulations, high order schemes allow to efficiently resolve the rapid small scale spatial oscillations of the solution and allow to alleviate the pollution effect [2, 3].

Mixed finite elements are extensively used to model electromagnetic wave propagation problems, and several high order extensions have been proposed in the literature [4, 5, 6, 7, 8]. As for standard Lagrange elements, though, the computational cost of solving the linear system of equations rapidly becomes overshadowed by the cost of assembling the finite element matrix as the order of the basis functions increases [9]. The aim of this paper is to solve this problem by reformulating the assembly procedure into a computationally more efficient procedure.

The paper is organized as follows. In section 2 we start by reviewing the classical assembly technique and by highlighting the most time consuming part. We then propose in section 3 a reformulation of the assembly procedure in terms of computationally efficient matrix-matrix products. We also address in section 4 the basis orientation problem. We conclude by presenting numerical results for a three dimensional wave propagation problem in section 5.

2 Classical finite element assembly procedure

Let us start by considering the time harmonic behavior of the electric field $\mathbf{e}(x,y,z)$ in a non-dissipative bounded domain $\Omega(x,y,z)$, with Dirichlet boundary conditions on its boundary. From Maxwell's equations, the electric field can be obtained by solving the following weak formulation [10]: find $\mathbf{e}(x,y,z) \in H_e(\mathbf{curl},\Omega)$ such that:

$$\int_\Omega \Big(\boldsymbol{\nu}_r \,\mathbf{curl}\,\mathbf{e}\Big)^T \Big(\,\mathbf{curl}\,\mathbf{f}^j\Big)\mathrm{d}\Omega \;- k^2 \int_\Omega \Big(\boldsymbol{\varepsilon}_r\mathbf{e}\Big)^T \Big(\mathbf{f}^j\Big)\mathrm{d}\Omega = 0, \qquad \forall \mathbf{f}^j \in H_0(\mathbf{curl},\Omega), \tag{1}$$

where $\boldsymbol{\nu}_r(x,y,z)$ is the relative reluctivity tensor, $\boldsymbol{\varepsilon}_r(x,y,z)$ is the relative electric permittivity tensor, and k is the wavenumber. The non-homogeneous (resp. homogeneous) Dirichlet boundary condition is taken into account in the curl-conforming vector space $H_e(\mathbf{curl},\Omega)$ (resp. $H_0(\mathbf{curl},\Omega)$). All the developments hereafter could be applied in a more general setting (*e.g.* with source terms, dissipation or different boundary conditions), but we consider (1) for simplicity.

Then, two levels of discretization are applied. First, the studied domain $\Omega(x,y,z)$ is meshed into simple elements $\Omega^e(x,y,z)$ (triangles or quadrangles in 2D; tetrahedra, hexahedra, prisms or pyramids in 3D). Moreover, a mapping between each mesh element $\Omega^e(x,y,z)$ and a reference element $\omega(u,v,w)$ is defined with an associated Jacobian matrix $\mathbf{J}^e(u,v,w)$. Finally, the solution $\mathbf{e}(x,y,z)$ is approximated in a finite dimensional curl-conforming[1] polynomial function space $\mathbb{B}(\omega)$, piece-wise defined on the elements [4, 5, 6, 7, 8]. The size of this space increases with the order of the polynomial approximation.

The matrix form of the resulting discretized harmonic problem can then be obtained by computing the following elementary terms [10]:

$$\mathcal{T}^e_{i,j} = \mathcal{I}^e_{i,j} - k^2\,\mathcal{K}^e_{i,j}, \tag{2}$$

where

$$\left\{\begin{array}{rcl} \mathcal{I}^e_{i,j} & = & \displaystyle\int_\omega \left(\boldsymbol{\nu}_r \frac{\mathbf{J}^e}{\det \mathbf{J}^e}\,\mathbf{curl}\,\mathbf{f}^i\right)^T \left(\frac{\mathbf{J}^e}{\det \mathbf{J}^e}\,\mathbf{curl}\,\mathbf{f}^j\right) \det \mathbf{J}^e \,\mathrm{d}\omega, \\ \mathcal{K}^e_{i,j} & = & \displaystyle\int_\omega \Big(\boldsymbol{\varepsilon}_r\,\mathbf{J}^{e^{-T}}\mathbf{f}^i\Big)^T \Big(\mathbf{J}^{e^{-T}}\mathbf{f}^j\Big) \det \mathbf{J}^e \,\mathrm{d}\omega, \end{array}\right. \tag{3}$$

with $\mathbf{f}^i, \mathbf{f}^j \in \mathbb{B}(\omega)$. Each $\mathcal{T}^e_{i,j}$ is giving the contribution of the degree of freedom (DOF) i and j of the mesh element Ω^e.

With these expressions, the classical finite element (FE) assembly algorithm is quite simple. It consists in iterating on every mesh element, computing the integrals (3) for every pair of DOFs i and j using a numerical quadrature rule, and adding the result in a global finite element matrix. Figure 1 illustrates this procedure in pseudo code.

It is worth noticing that the impact on the computation time of increasing the basis order is twofold:

1. each element will have more DOFs, thus increasing the number of $\mathcal{T}^e_{i,j}$ to compute;

[1] Note that all what follows can be easily applied to other kinds of high order Whitney forms, *i.e.* standard conforming elements, div-conforming elements or discontinuous elements. The curl-conforming case exhibits all the important properties of the general assembly procedure.

```
for e in [1, size(element)]
  dof[] = takeDof(e);

  for i in [1, size(dof)]
    for j in [1, size(dof)]
      globalI = globalId(dof[i]);
      globalJ = globalId(dof[j]);

      // A: Finite Element Global Matrix //
      A[globalI, globalJ] += feTerm(e, i, j);
```

```
feTerm(e, i, j)
  return quadrature(I^e_{i,j}) - k^2 * quadrature(K^e_{i,j});
```

Figure 1: Classical assembly algorithm.

2. the numerical quadrature will require more points, thus slowing down the evaluation of each $\mathcal{T}^e_{i,j}$

These two phenomena will substantially increase the assembly time of the FE linear system. To give some order of magnitude, Table 1 shows the number of $\mathcal{K}^e_{i,j}$ terms needed for a tetrahedral finite element.

Curl-conforming basis order	1	2	3	4
Number of integration points	4	11	24	43
Number of $\mathcal{K}^e_{i,j}$ terms	144	900	3600	11025

Table 1: Number of $\mathcal{K}^e_{i,j}$ terms and required integration points for a tetrahedral finite element.

3 Efficient assembly procedure using matrix operations

The key idea of a fast assembly procedure is to compute all the $\mathcal{T}^e_{i,j}$ terms using matrix-matrix products, as proposed in [9, 11] for discontinuous Galerkin schemes using high order nodal Lagrange elements.

Indeed, matrix operations exhibit an excellent cache reuse, and are standardized in the Basic Linear Algebra Subprograms (BLAS) interface [12, 13]. Moreover, highly optimized and parallel implementations are available for modern multi-core architectures (*e.g.* MKL [14], OpenBLAS [15], ATLAS [16]).

In order to use this approach, the integrals of (3) have to be rewritten. These developments are detailed below.

3.1 Matrix version of $\mathcal{I}^e_{i,j}$

Let us start by simplifying the $\mathcal{I}^e_{i,j}$ integral:

$$\begin{aligned}\mathcal{I}^e_{i,j} &= \int_\omega \left(\boldsymbol{\nu}_r \frac{\mathbf{J}^e}{\det \mathbf{J}^e}\,\mathbf{curl}\,\mathbf{f}^i\right)^T \left(\frac{\mathbf{J}^e}{\det \mathbf{J}^e}\,\mathbf{curl}\,\mathbf{f}^j\right) \det \mathbf{J}^e \,\mathrm{d}\omega,\\ &= \int_\omega \frac{1}{\det \mathbf{J}^e}\left(\underbrace{\boldsymbol{\nu}_r \mathbf{J}^e}_{\mathbf{K}^e}\,\mathbf{curl}\,\mathbf{f}^i\right)^T \left(\mathbf{J}^e\,\mathbf{curl}\,\mathbf{f}^j\right)\mathrm{d}\omega.\end{aligned}$$

For convenience, the product of the reluctivity tensor $\boldsymbol{\nu}_r$ and the Jacobian matrix $\mathbf{J}^e$ is defined as the matrix $\mathbf{K}^e$. Now, let us express the matrix-vector products in a per element fashion. The notation $A_{a,b}$ denotes the element at row number a and column number b of matrix $\mathbf{A}$, and the notation b_a refers to the element a of vector $\mathbf{b}$. Matrices are of dimension 3×3 and vectors of dimension 3×1. We have:

$$\begin{aligned}\mathcal{I}^e_{i,j} &= \int_\omega \frac{1}{\det \mathbf{J}^e} \sum_{a=1}^{3} \left(\sum_{b=1}^{3} K^e_{a,b}\,\mathrm{curl}_b\,\mathbf{f}^i\right)\left(\sum_{c=1}^{3} J^e_{a,c}\,\mathrm{curl}_c\,\mathbf{f}^j\right)\mathrm{d}\omega,\\ &= \int_\omega \sum_{a=1}^{3}\sum_{b=1}^{3}\sum_{c=1}^{3} \frac{1}{\det \mathbf{J}^e} K^e_{a,b}\,J^e_{a,c}\,\mathrm{curl}_b\,\mathbf{f}^i\,\mathrm{curl}_c\,\mathbf{f}^j\,\mathrm{d}\omega.\end{aligned}$$

Then, the integral can be computed using a numerical quadrature. The notation w_g denotes the g^{th} integration weight, and the notation $\alpha\big|_g$ means that α is evaluated at the g^{th} integration point. The total number of integration point is denoted by G. We get:

$$\begin{aligned}\mathcal{I}^e_{i,j} &= \sum_{g=1}^{G} w_g \sum_{a=1}^{3}\sum_{b=1}^{3}\sum_{c=1}^{3} \frac{1}{\det \mathbf{J}^e\big|_g} K^e_{a,b}\Big|_g J^e_{a,c}\Big|_g\,\mathrm{curl}_b\,\mathbf{f}^i\Big|_g\,\mathrm{curl}_c\,\mathbf{f}^j\Big|_g,\\ &= \sum_{g=1}^{G}\sum_{b=1}^{3}\sum_{c=1}^{3}\left(\frac{1}{\det \mathbf{J}^e\big|_g}\sum_{a=1}^{3} K^e_{a,b}\Big|_g J^e_{a,c}\Big|_g\right)\left(w_g\,\mathrm{curl}_b\,\mathbf{f}^i\Big|_g\,\mathrm{curl}_c\,\mathbf{f}^j\Big|_g\right),\\ &= \sum_{g=1}^{G}\sum_{b=1}^{3}\sum_{c=1}^{3}\mathbf{B}\Big[e,\{g,b,c\}\Big]\mathbf{C}\Big[\{g,b,c\},\{i,j\}\Big],\\ &:= \mathbf{D}\Big[e,\{i,j\}\Big],\end{aligned} \tag{4}$$

where the operations $\{\cdot,\cdot\}$ and $\{\cdot,\cdot,\cdot\}$ take two (or three) indices and combine them into a new unique one. Figure 2 illustrates this operation.

From equation (4), we can notice that each $\mathcal{I}^e_{i,j}$ term is an element of a matrix $\mathbf{D}$ at line e and column $\{i,j\}$. Moreover, this matrix $\mathbf{D}$ is constructed by the product of two other

matrices, **B** and **C**, defined as follow:

$$\begin{cases} \mathbf{B}\Big[e, \{g, b, c\}\Big] & = & \dfrac{1}{\det \mathbf{J}^e\Big|_g} \displaystyle\sum_{a=1}^{3} K^e_{a,b}\Big|_g \, J^e_{a,c}\Big|_g, \\ \\ \mathbf{C}\Big[\{g, b, c\}, \{i, j\}\Big] & = & w_g \operatorname{curl}_b \mathbf{f}^i\Big|_g \operatorname{curl}_c \mathbf{f}^j\Big|_g. \end{cases} \tag{5}$$

It is worth noticing that matrix **B** is storing only mesh (*i.e.* metric) dependent data, while matrix **C** does not. The latter is defined only on the reference space ω.

Figure 2 illustrates the structure of matrices **B** and **C**.

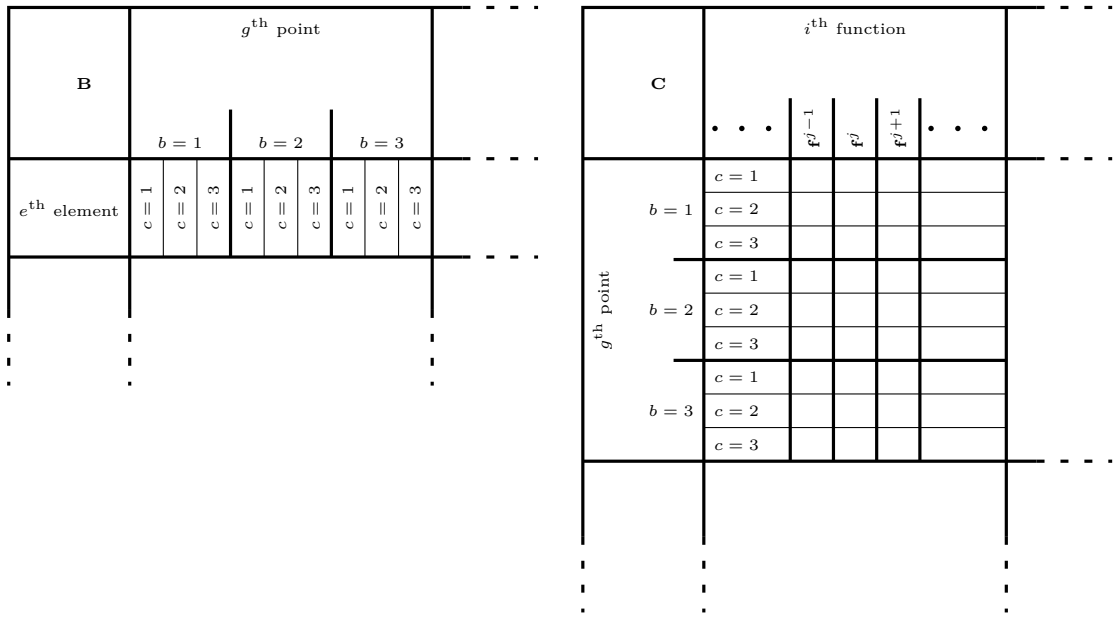


Figure 2: Structure of matrices **B** and **C** for the computation of $\mathcal{I}^e_{i,j}$.

3.2 Matrix version of $\mathcal{K}^e_{i,j}$

We can apply the same procedure with the $\mathcal{K}^e_{i,j}$ integral. As previously, we start by defining a new matrix $\mathbf{L}^{e^{-T}}$ as the product of the relative permittivity tensor and the inverted, transposed Jacobian matrix. Then, the matrix-vector products are expressed in a per element

way:

$$\begin{aligned}
\mathcal{K}_{i,j}^{e} &= \int_{\omega} \left(\underbrace{\boldsymbol{\varepsilon}_r \, \mathbf{J}^{e^{-T}}}_{\mathbf{L}^{e^{-T}}} \mathbf{f}^i \right)^T \left(\mathbf{J}^{e^{-T}} \mathbf{f}^j \right) \det \mathbf{J}^e \, \mathrm{d}\omega, \\
&= \int_{\omega} \sum_{a=1}^{3} \left(\sum_{b=1}^{3} L_{a,b}^{e^{-T}} \, f_b^i \right) \left(\sum_{c=1}^{3} J_{a,c}^{e^{-T}} \, f_c^j \right) \det \mathbf{J}^e \, \mathrm{d}\omega, \\
&= \int_{\omega} \sum_{a=1}^{3} \sum_{b=1}^{3} \sum_{c=1}^{3} L_{a,b}^{e^{-T}} \, J_{a,c}^{e^{-T}} \, f_b^i \, f_c^j \, \det \mathbf{J}^e \, \mathrm{d}\omega.
\end{aligned}$$

This integral is evaluated using a numerical quadrature:

$$\begin{aligned}
\mathcal{K}_{i,j}^{e} &= \sum_{g=1}^{G} w_g \sum_{a=1}^{3} \sum_{b=1}^{3} \sum_{c=1}^{3} L_{a,b}^{e^{-T}} \Big|_g J_{a,c}^{e^{-T}} \Big|_g f_b^i \Big|_g f_c^j \Big|_g \det \mathbf{J}^e \Big|_g, \\
&= \sum_{g=1}^{G} \sum_{b=1}^{3} \sum_{c=1}^{3} \left(\det \mathbf{J}^e \Big|_g \sum_{a=1}^{3} L_{a,b}^{e^{-T}} \Big|_g J_{a,c}^{e^{-T}} \Big|_g \right) \left(w_g \, f_b^i \Big|_g f_c^j \Big|_g \right), \\
&= \sum_{g=1}^{G} \sum_{b=1}^{3} \sum_{c=1}^{3} \mathbf{E}\Big[e, \{g,b,c\}\Big] \mathbf{F}\Big[\{g,b,c\}, \{i,j\}\Big], \\
&:= \mathbf{G}\Big[e, \{i,j\}\Big].
\end{aligned} \tag{6}$$

From equation (6) we recover a result analog to equation (4): each $\mathcal{K}_{i,j}^{e}$ is an element at line e and column $\{i,j\}$ of a matrix $\mathbf{G}$. This matrix is constructed by the product of $\mathbf{E}$ and $\mathbf{F}$, defined as follow:

$$\begin{cases}
\mathbf{E}\Big[e, \{g,b,c\}\Big] &= \det \mathbf{J}^e \Big|_g \sum_{a=1}^{3} L_{a,b}^{e^{-T}} \Big|_g J_{a,c}^{e^{-T}} \Big|_g, \\
\mathbf{F}\Big[\{g,b,c\}, \{i,j\}\Big] &= w_g \, f_b^i \Big|_g f_c^j \Big|_g.
\end{cases} \tag{7}$$

Finally, as for the $\mathcal{I}_{i,j}^{e}$ term, one matrix is mesh dependent and the other is not.

3.3 Efficient assembly algorithm

With the matrix computations of the elementary integrals $\mathcal{I}_{i,j}^{e}$ and $\mathcal{K}_{i,j}^{e}$, we can now propose a new assembly algorithm as displayed on Figure 3. The main difference compared to the classical algorithm is the implementation of `feTerm()`: in the classical approach it was computing the integrals $\mathcal{I}_{i,j}^{e}$ and $\mathcal{K}_{i,j}^{e}$ on the fly, while in the new approach it fetches the precomputed integrals. It is worth mentioning that the construction of the metric-dependent matrices requires a loop on every mesh element.

```
B[][] = computeB(); // Requires a loop on every mesh element
C[][] = computeC(); // Does not require a loop on every mesh element
D[][] = B * C;

E[][] = computeE(); // Requires a loop on every mesh element
F[][] = computeF(); // Does not require a loop on every mesh element
G[][] = E * F;

for e in [1, size(element)]
  dof[] = takeDof(e);

  for i in [1, size(dof)]
    for j in [1, size(dof)]
      globalI = globalId(dof[i]);
      globalJ = globalId(dof[j]);

      // A: Finite Element Global Matrix //
      A[globalI, globalJ] += feTerm(e, i, j);
```

```
feTerm(e, i, j)
  return D[e][combineIndex(i, j)] - k^2 * G[e][combineIndex(i, j)];
```

Figure 3: New assembly algorithm.

4 Global orientation of the basis functions

As showed in section 3, the matrices **C** and **F** in, respectively, equations (5) and (7), are defined by the basis functions in the reference element. Thus we need to know the basis functions *before* iterating on the mesh elements. With classical Lagrange basis functions, this doesn't lead to any problem, since the basis functions in the reference element do not depend in any way on the mesh elements.

However, this is not the case with more general basis functions. Let us compare for instance a Lagrange basis and a Legendre basis on a one dimensional line element. The two bases are of order 3. Figure 4 illustrates the two edge-based functions of each basis. This figure shows also two situations: in the first one, the line element is oriented from *left to right*, and from *right to left* in the second case. It can be directly seen from Figure 4 that the Lagrange basis functions can be *permuted* depending on the line orientation, while the Legendre functions cannot. It can be argued that the Legendre functions can be multiplied by -1 depending on the parity and the line orientation: unfortunately, this is not true for the three dimensional case [17].

With the classical assembly procedure, this orientation problem can be easily solved: when iterating on the elements, the edges and faces orientations can be analysed on the fly. Then, the correctly oriented basis functions can be chosen and fed to the `feTerm()` function [18]. However, with the matrix approach, the basis functions used in **C** and **F** of (5) and (7) have to be known *before* iterating on the elements. Moreover, they can be different from element to element when using non-Lagrange functions.

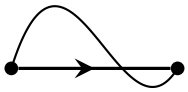
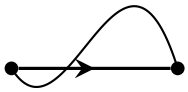
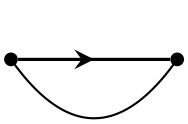
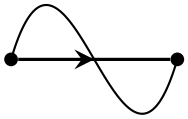
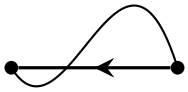
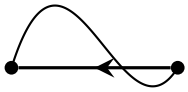
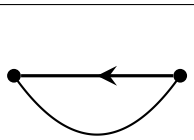
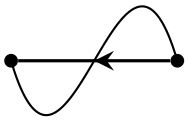
Orientation	Lagrange		Legendre	
	f_0	f_1	f_0	f_1
Left – Right				
Right – Left				

Figure 4: A one dimensional comparison of an ordre 3 Lagrange basis and Legendre basis (edge contribution only).

To overcome this situation, we need to sort the elements with respect to the orientation of their edges and faces. Then, a *set* of matrices **B**, **C**, **D**, **E**, **F** and **G** can be computed for every group of elements sharing the same orientation, and thus the same oriented basis functions. A general and automatic procedure to handle the orientation is proposed hereafter.

4.1 Permutation tree to represent orientations

We assume that every vertex of the mesh is given a unique number, called vertex number. A mesh element is represented by an *ordered* vertex number sequence, *i.e.* a vector of vertex numbers. For instance, the sequence $\mathbf{s} = [42, 17, 16, 21]$ corresponds to the mesh element composed by the vertices 42, 17, 16 and 21.

The orientation of a mesh element is specified through its vertex numbers. For instance, the positive orientation of an edge is conventionally from the vertex with the smallest number towards the vertex with the largest number. Practical orientation rules (based on vertex numbers) for edges, triangular faces and quadrangular faces can be found in [8].

Let us consider the simplest case of a mesh element Ω^e composed of V vertices in the set $\mathcal{V} = \{0, 1, \ldots, V-1\}$. This particular element is then one of the $V!$ possible permutations of the set $\mathcal{V}$. A tree structure can be used to represent those permutations. This permutation tree has the following properties:

1. the tree has a depth of V;
2. a node at depth d has $V - d$ children[2];
3. each node has a label between 0 and $V - 1$, with the exception of the root, that stays unlabeled;
4. sibling nodes have strictly different labels;
5. a node cannot be given a label already taken by one of its ancestors;

[2]The depth of the root node is equal to 0.

6. the descendants of a node are sorted with increasing labels;

7. a leaf can be tagged.

With this structure, the ordered sequence of labels on a path from the root to a leaf is a possible permutation of the set $\mathcal{V}$. Moreover, this structure has exactly $V!$ possible paths that leads to different label sequences. Thus, this tree spans all the possible permutations of the set $\mathcal{V}$.

This tree structure can be also used as a permutation *dictionary*: a given permuted sequence can be matched to a path in the tree, and thus to a leaf tag. A sequence $\mathbf{s}$ corresponds to the path traveling (in order) through the nodes labeled $\mathbf{s}(1)$, $\mathbf{s}(2)$, ..., $\mathbf{s}(V)$. Moreover, using property 7, the ending node of the path can be associated to a tag. Thus, this structure is a dictionary, where a permuted sequence is a key that maps to a user-defined value. The matching time scales in $V \log_2 V$. Indeed, we need to go through the V nodes of a path. And at each level, the next node can be found by a dichotomic search that scales in $\log_2 V$. This dichotomic approach is allowed thanks to property number 6. Figure 5 illustrates the permutation tree structure in the quadrangular case: that is with $\mathcal{V} = \{0, 1, 2, 3\}$.

4.2 Generation of all possible orientations

The permutation tree structure can be used to generate every possible orientation. The simplest solution is to consider each vertex permutation as a new orientation. However, the number of orientations will grow as $V!$. For instance, with this strategy, 40320 (*i.e.* 8!) orientations must be considered for a hexahedron.

To overcome this factorial growth, mappings can be computed between vertex permutations sharing the same edge connectivity: this leads to a splitting of the permutations into connectivity groups. Only one member of each group is then required to define the orientation, since the other members can be mapped onto the reference one (through univoque, and thus invertible, map). The orientation algorithm then reduces to:

1. generate the permutation tree associated to an element type (*e.g.* triangle, hexahedron, ...);

2. group every vertex permutations into connectivity groups;

3. give a unique tag to every group;

4. tag every leaf with the connectivity group associated to the path ending at this node;

5. select one permutation (*i.e.* one leaf of the permutation tree) in every connectivity group as reference.

It is worth noticing that step 2 is time consuming, since it requires to compare the connectivity of all the vertex permutations. However, it only needs to be applied (offline) once per element type, and the resulting tree can be reused.

Figure 5 illustrates a permutation tree for a quadrangular element type, that has been tagged with its connectivity groups. Figure 6 shows the oriented reference permutations

proposed by the tree. It is worth noticing that it is impossible to map these quadrangles on each other.

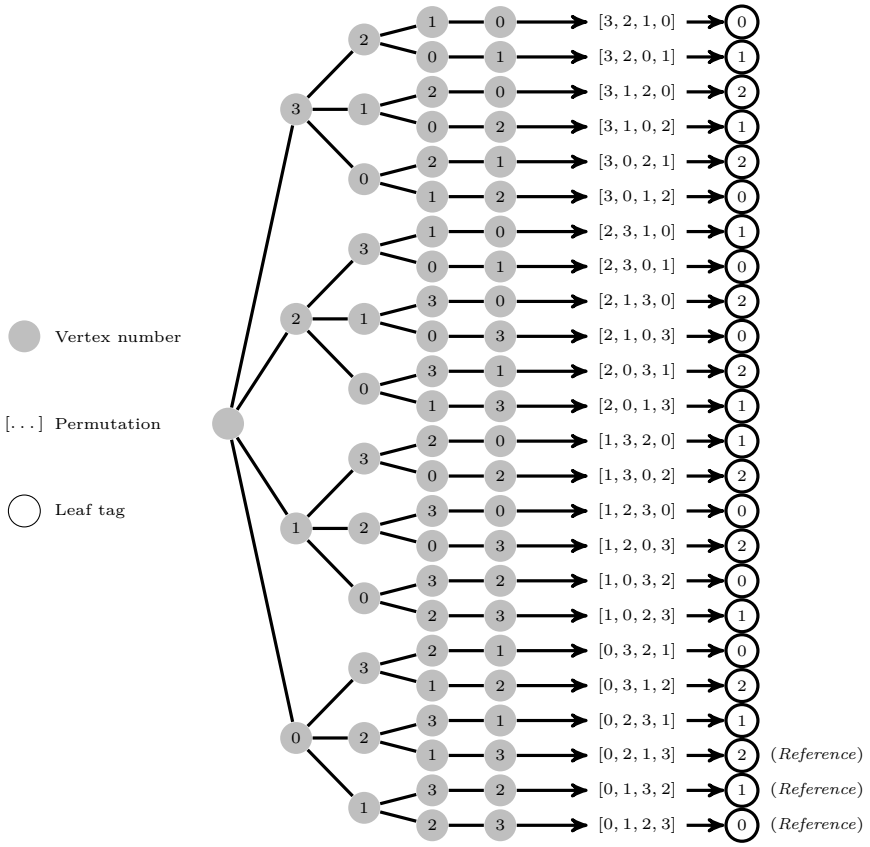


Figure 5: Permutation tree for the quadrangular case.

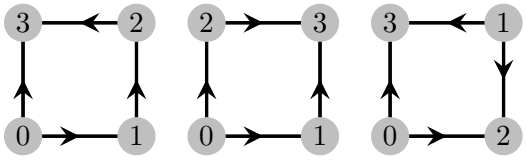


Figure 6: The three possible orientations for a quadrangle, as proposed by the tree of Figure 5.

4.3 Finding the orientation associated to a mesh element

Once a permutation tree is available for each type of element present in the mesh, every mesh element can be mapped to one connectivity group. Indeed, the orientation rules depend only on the bigger/smaller relations among the vertex numbers of a mesh element, and not on their actual value. Thus, from any mesh element, it is possible to find an equivalent element composed by vertex numbers in $\mathcal{V}$. Doing so, the orientation of a mesh element can be found efficiently. Algorithmicaly, we get:

1. get an equivalent (from the orientation point of view) element composed by vertex numbers in $\mathcal{V}$;
2. use the permutation tree to match the new vertex number sequence with a leaf tag;
3. the connectivity tag of the leaf defines the orientation of the given mesh element;
4. map the mesh element onto the reference member of the connectivity group.

It is also worth remembering that the matching procedure scales in $V \log_2 V$.

For each connectivity group and element type, it is then possible to compute the matrices **C** and **F**. Matrices **B** and **E** are metric-dependent, and contain per-element entries, for each connectivity group and element type. The Jacobian matrices involved in the computation of **B** and **E** are obtained by composition of two mappings: one from the actual connectivity group member to the reference member, and one from the reference member to the actual mesh element.

Table 2 shows the number of possible orientations for different element types.

Element type	Number of vertices	Number of possible orientations
Line	2	1
Triangle	3	1
Quadrangle	4	3
Tetrahedron	4	1
Pyramid	5	15
Prism	6	60
Hexahedron	8	840

Table 2: Number of possible orientations for common element types.

5 Results

In this section, a comparison of the classical FE assembly approach and the matrix approach is presented. Practically, two codes are compared. These two versions are sharing the same implementation, except for the function `feTerm()`: one is using the classical version presented on Figure 1, and the other is using the new version on Figure 3.

The test case consists in a propagation problem into a curved wave guide meshed with 6950 curved tetrahedra. A constant source is imposed at one end, and a perfect reflector at the other end. The simulations are run on a Intel Xeon E5645 with 6 threads for both the classical and the matrix versions. The latter uses the OpenBLAS [15] implementation of BLAS. The assembly time is recorded for an increasing basis order. The results are depicted on Figure 7. It can be directly seen that the new proposed assembly procedure outperforms the classical one, and that the speedup increases with the order of the basis functions. For instance, for 6-th order basis functions (1097824 unknowns), the speedup reaches 18.

An other interesting comparison is the ratio between the time spend assembling the matrix and the time spent solving the FE linear system. The latter is computed using the direct

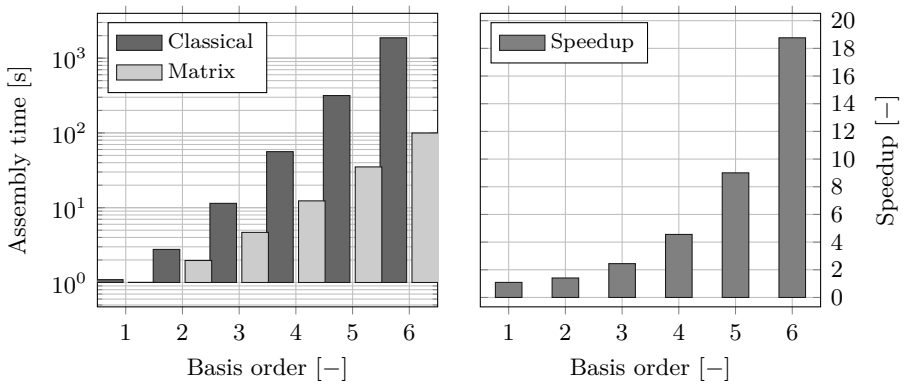


Figure 7: Assembly time and speedup for the classical and matrix assembly procedures.

solver MUMPS [19, 20] with 1 MPI process and 6 threads. Figure 8 displays the two timings with 4-th order basis functions (386960 unknowns). It is worth noticing that for the classical

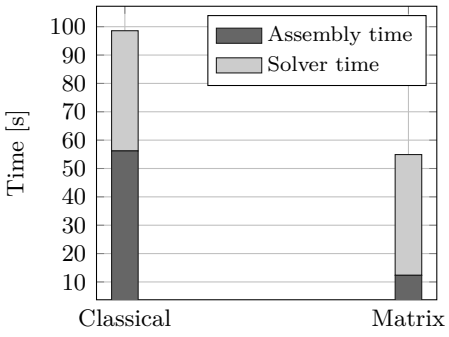


Figure 8: Time spent assembling the FE matrix and solving the FE linear system for the two assembly approaches using 4-th order basis functions.

assembly technique, the time spent assembling the FE system exceeds the time required to solve it. This not the case anymore using the matrix approach.

One last test is the memory impact of the two versions. If we compare the maximum amount of resident memory needed by the two implementations, we end up with relatively close values. Actually, in order to have a fair comparison, the two implementations are pre-evaluating and storing the Jacobian matrices. This explains the above results, since the Jacobian matrices are dominating the memory impact. To give some order of magnitude, with 6-th order basis functions, the maximum resident memory needed by the classical version is 21 GB and 28 GB for the matrix version.

6 Conclusion

In this work, we presented an efficient finite element assembly procedure for high order Whitney forms, and in particular high order curl-conforming elements. This objective was reached by using computationally efficient matrix-matrix products. Since non-Lagrange elements are

used, the orientation of the vector-valued basis functions was treated. Moreover, the classical on the fly orienting procedure is not compatible with the matrix-matrix assembly. Thus an *a priori* solution was developed. A computationally efficient permutation tree structure enables the implementation of an orientation dictionary. This approach leads to both the generation of all possible orientations, and a fast matching between a mesh element and its orientation.

Acknowledgments

This work was supported in part by the Belgian Science Policy Office under grant IAP P7/02 and the Walloon Region under grant WIST3 DOMHEX. N. Marsic is a fellowship beneficiary with the Belgian Research Training Fund for Industry and Agriculture (FRIA).

References

[1] P. E. Vincent and A. Jameson, "Facilitating the adoption of unstructured high-order methods amongst a wider community of fluid dynamicists," *Mathematical Modelling of Natural Phenomena*, vol. 6, pp. 97–140, 2011.

[2] F. Ihlenburg and I. Babuška, "Finite element solution of the Helmholtz equation with high wave number part I: The h-version of the FEM," *Computers & Mathematics with Applications*, vol. 30, no. 9, pp. 9 – 37, 1995.

[3] F. Ihlenburg and I. Babuška, "Finite element solution of the Helmholtz equation with high wave number part II: The h-p version of the FEM," *SIAM Journal on Numerical Analysis*, vol. 34, no. 1, pp. 315–358, 1997.

[4] J. P. Webb and B. Forghani, "Hierarchal scalar and vector tetrahedra," *IEEE Transactions on Magnetics*, vol. 29, no. 2, pp. 1495–1498, 1993.

[5] T. V. Yioultsis and T. D. Tsiboukis, "A fully explicit Whitney element-time domain scheme with higher order vector finite elements for three-dimensional high frequency problems," *IEEE Transactions on Magnetics*, vol. 34, no. 5, pp. 3288–3291, 1998.

[6] C. Geuzaine, "High order hybrid finite element schemes for Maxwell's equations taking thin structures and global quantities into account," PhD thesis, Université de Liège, Belgium, Dec. 2001.

[7] A. Bossavit, "Generating Whitney forms of polynomial degree one and higher," *IEEE Transactions on Magnetics*, vol. 38, pp. 341–344, Mar 2002.

[8] S. Zaglmayr, "High order finite element methods for electromagnetic field computation," PhD thesis, Johannes Kepler Universität, Austria, Aug. 2006.

[9] K. Hillewaert, "Exploiting data locality in the DGM discretisation for optimal efficiency," in *ADIGMA - A European Initiative on the Development of Adaptive Higher-Order Variational Methods for Aerospace Applications* (N. Kroll, H. Bieler, H. Deconinck, V. Couaillier, H. Ven, and K. Sørensen, eds.), vol. 113 of *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, pp. 11–23, Springer Berlin Heidelberg, 2010.

[10] A. Bossavit, *Computational Electromagnetism: Variational Formulations, Complementarity, Edge Elements*. Academic Press, 1998.

[11] J. Lambrechts, "Finite element methods for coastal flows: Application to the great barrier reef," PhD thesis, Université catholique de Louvain, Belgium, Mar. 2011.

[12] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley, "An updated set of Basic Linear Algebra Subprograms (BLAS)," *ACM Transactions on Mathematical Software*, vol. 28, pp. 135–151, June 2002.

[13] B. Kågström, P. Ling, and C. V. Loan, "GEMM-based level 3 BLAS: high-performance model implementations and performance evaluation benchmark," *ACM Transactions on Mathematical Software*, vol. 24, pp. 268–302, Sept. 1998.

[14] Intel Corporation, *Reference Manual for Intel Math Kernel Library.*

[15] Z. Xianyi, W. Qian, and Z. Yunquan, "Model-driven level 3 BLAS performance optimization on Loongson 3A processor," in *Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on*, pp. 684–691, Dec. 2012.

[16] R. C. Whaley and A. Petitet, "Minimizing development and maintenance costs in supporting persistently optimized BLAS," *Software: Practice and Experience*, vol. 35, pp. 101–121, Feb. 2005.

[17] M. Ainsworth and J. Coyle, "Hierarchic finite element bases on unstructured tetrahedral meshes," *International Journal for Numerical Methods in Engineering*, vol. 58, no. 14, pp. 2103–2130, 2003.

[18] P. Dular, C. Geuzaine, F. Henrotte, and W. Legros, "A general environment for the treatment of discrete problems and its application to the finite element method," *IEEE Transactions on Magnetics*, vol. 34, pp. 3395–3398, Sept. 1998.

[19] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent, "A fully asynchronous multifrontal solver using distributed dynamic scheduling," *SIAM Journal on Matrix Analysis and Applications*, vol. 23, no. 1, pp. 15–41, 2001.

[20] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet, "Hybrid scheduling for the parallel solution of linear systems," *Parallel Computing*, vol. 32, no. 2, pp. 136–156, 2006.