Higher-Order Discontinuous Galerkin Method for Pyramidal Elements using Orthogonal Bases

Morgane Bergot, Marc Duruflé

To cite this version:


HAL Id: hal-00547319
https://hal.archives-ouvertes.fr/hal-00547319v2
Submitted on 13 Dec 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
HIGHER-ORDER DISCONTINUOUS GALERKIN METHOD FOR PYRAMIDAL ELEMENTS USING ORTHOGONAL BASES

Morgane Bergot
Project POems, INRIA Rocquencourt, Le Chesnay, France
Email: morgane.bergot@inria.fr

Marc Duruflé
Institut Mathématique de Bordeaux, Université Bordeaux I, Bordeaux, France
Email: marc.duruflé@math.u-bordeaux1.fr

Mathematics subject classification: 65F05, 65F50, 65M60, 65Y20

Abstract

We study finite elements of arbitrarily high-order defined on pyramids for discontinuous Galerkin methods. We propose a new family of high-order pyramidal finite elements using orthogonal basis functions which can be used in hybrid meshes including hexahedra, tetrahedra, wedges and pyramids. We perform a comparison between these orthogonal functions and nodal functions for affine and non-affine elements. Different strategies for the inversion of the mass matrix are also considered and discussed. Numerical experiments are conducted for the 3-D Maxwell’s equations.

Key words: pyramidal element, higher-order finite element, hybrid mesh, conformal mesh, discontinuous Galerkin method, orthogonal basis functions.

Introduction

Discontinuous Galerkin methods have been extensively studied for tetrahedral meshes (e.g. Hesthaven and Warburton [1] for Maxwell’s equations). The works of Cohen and collaborators (Pernet and Ferrières [2], Duruflé [3]) have shown the higher efficiency obtained by using hexahedral meshes with this method, thanks to the tensorization of the basis functions. However, generating high quality conformal or non-conformal hexahedral meshes is not an easy task.

A first solution is to consider a non-conformal mesh with only tetrahedra and hexahedra, some quadrilateral faces connecting an hexahedron with two tetrahedra. Even though attractive, this method requires to handle non-conformal meshes and may not be the most efficient. A second solution is to allow the insertion of other types of elements, pyramids and prisms, in order to make a conformal transition between triangular and quadrilateral faces. As pyramidal elements are not very common in the Discontinuous Galerkin literature, it seemed appealing to explore the second option. This second option is the only solution studied in this paper.

Pyramidal elements have been studied by several authors in the context of continuous finite elements (Zaghmaya cited by Demkowicz [4], Nigam and Phillips [5] [6], Bergot et al. [7], Sherwin et al. [8]) but their application to discontinuous Galerkin methods has received less attention. Obviously, basis functions developed for continuous finite elements can be used for discontinuous elements as done in [7]. However, since continuity is not required, other sets of basis functions are acceptable and may have better properties. An attractive choice consists of orthogonal bases, which has been proposed by Kirby et al. [9] for all the types of elements,
but the proposed basis functions for pyramids generate \( P_r \) instead of the optimal finite element space, and we illustrate here that the use of \( P_r \) for pyramids provides a poor convergence for non-affine pyramids. A recent work of Gassner et al. [10] proposes an original approach to construct nodal discontinuous Galerkin method for hybrid meshes, since it avoids the use of a reference element, by constructing directly nodal functions generating polynomials on the real element.

We consider linear hyperbolic problems like Maxwell’s equations, with an explicit time scheme like Runge-Kutta. Continuous Galerkin finite element schemes are not attractive in this context, since the mass matrix is large and costly to invert. Mass-lumped elements are well-known for hexahedra (Cohen [11]) but less so for other elements (for tetrahedra see Mulder et al. [12]) that require a large number of additional degrees of freedom and lead to a more restrictive stability condition. Furthermore, the application of a continuous Galerkin formulation is tedious because of spurious modes: it often needs regularization, as described by Costabel in [13] for Maxwell’s equations. When a discontinuous Galerkin method is considered, nonetheless we get a block-diagonal mass matrix, but orthogonal tensorized basis functions can be used to get an elementary sparse mass matrix, which induces a gain in computational time when the order of approximation is large enough.

In this paper, we propose finite elements that reduce both memory and computational time as much as possible. We propose two sets of basis functions based on orthogonal tensorized basis functions: a set generating optimal finite element spaces in the sense of the convergence defined in [7] used with non-affine hexahedra, prisms, pyramids, and a set generating polynomials \( P_r \) used with affine elements.

The outline of our paper is as follows:

- We present the representative problem we study and its discretization in space and in time in Section 1.
- We detail the construction of the mass matrix for pyramidal elements in Section 2, when using classical nodal basis functions in Section 2.1 and orthogonal basis functions in Section 2.2. The mass-matrix is constructed and factorised only once, but a sparse structure of the matrix would speed-up the resolution phase performed at each time step. A way to obtain a diagonal mass matrix proposed by Warburton [14] is considered in Section 2.3.
- Section 3 is devoted to the construction of a fast matrix-vector product using orthogonal basis functions for all the types of elements. For pyramidal elements, its complexity is in \( O(r^4) \) where \( r \) is the order of approximation, instead of \( O(r^6) \) when no tensorization is available (e.g. nodal functions of the pyramid). Orthogonal functions are asymptotically more efficient than nodal functions, but it has been numerically found that it was more efficient for pyramids from \( r \geq 3 \), for wedges from \( r \geq 5 \) and for tetrahedra from \( r \geq 10 \).
- We perform numerical comparisons of the finite elements constructed to several other elements in Section 4. For affine elements, the orthogonal basis functions generating optimal finite element spaces are more efficient. A comparison of computational times shows that affine pyramids are more efficient than affine tetrahedra from order 7 on, prisms are always more efficient than tetrahedra, and non-affine pyramids are more efficient than curved tetrahedra sticked to a quadrangular face from order 4 and above. That is why considering prisms and pyramids instead of non-conformal meshes with tetrahedra is relevant.
• Numerical results are provided in Section 5 for the resolution of 3-D time-domain Maxwell’s equations.

1. Definitions and Presentation of the Problem

1.1. Definitions and Variational Formulation

We consider the following representative linear hyperbolic problem (e.g. Godlewski and Raviart [15])

\[
M \frac{\partial u}{\partial t} + \sum_{1 \leq i \leq d} A_i \frac{\partial u}{\partial x_i} + \sum_{1 \leq i \leq d} B_i \frac{\partial u}{\partial x_i} = 0, \quad (M, A_i, B_i) \in (M_{n_s} (\mathbb{R}))^3, \quad u \in \mathbb{R}^{n_s},
\]

where \(n_s\) is the number of scalar unknowns of the equation, and \(d\) is the dimension. When the system is symmetric, we have \(B_i = A_i^*\).

Remark 1.1. This formulation is useful to exhibit the antisymmetry of the stiffness matrix when using centered flux and without absorbing conditions, which is essential to get the stability.

Let \(\Omega\) be an open set of \(\mathbb{R}^3\), composed of \(n_e\) elements \(K_i\)

\[
\Omega = \bigcup_{1 \leq i \leq n_e} K_i.
\]

For any element \(K\) of boundary \(\partial K\) of outward normal \(n\), we consider a discontinuous method. For example, the Local Discontinuous Galerkin (LDG) formulation (see Hesthaven and Warburton [16] for Maxwell’s equations) writes as

\[
\begin{aligned}
\text{Find } u \in V \text{ such that } \\
\forall v \in V, \quad & \frac{d}{dt} \int_K M u \cdot v \, dx - \int_K \sum_{1 \leq i \leq d} \left( A_i u \cdot \frac{\partial v}{\partial x_i} - B_i \frac{\partial u}{\partial x_i} \cdot v \right) \, dx \\
&+ \int_{\partial K} (N_1 \{u\} + N_2 [u]) \cdot v \, ds = 0,
\end{aligned}
\]

where \(V = (L^2(\Omega))^{n_s}\), \(N_1 = \sum_{1 \leq i \leq d} A_i n_i\), \(N_2 = \sum_{1 \leq i \leq d} B_i n_i\). The average \(\{u\}\) is

\[
\{u\} = \frac{1}{2}(u_1 + u_2)
\]

and \([u]\) is

\[
[u] = \frac{1}{2}(u_2 - u_1) + \frac{1}{2} \alpha \int_{\partial K} C(u_2 - u_1) \, ds,
\]

where \(C\) is a symmetric positive matrix, \(u_1\) value of \(u\) on the element \(K\) and \(u_2\) value of \(u\) on a neighbour element of \(K\), and \(\alpha \leq 0\). In general, we take \(\alpha = -0.5\) in our experiments.
1.2. Space Discretization

Given a finite-dimensioned subspace $V_h$ of the space $V$, the discrete problem reads

\[
\begin{aligned}
\forall v_h \in V_h, \quad & \frac{d}{dt} \int_K M u_h \cdot v_h \, dx - \int_K \sum_{1 \leq i \leq d} \left( A_i u_h \frac{\partial v_h}{\partial x_i} - B_i \frac{\partial u_h}{\partial x_i} \cdot v_h \right) \, dx \\
& + \int_{\partial K} \left( N_1 \{u_h\} + N_2 [u_h]\right) \cdot v_h \, ds = 0.
\end{aligned}
\]  

(1.3)

We denote by $n = n_s n_r$ the dimension of $V_h$.

To write the integrals on the reference element, the standard transformation (Ciarlet [17]) $F$ from a reference element $\hat{K}$ to the mesh element $K$ is used for hexahedral, tetrahedral and wedge elements. For pyramidal elements, following Bedrosian [18], we define a transformation using rational fractions.

**Definition 1.1.** The transformation $F$ from the reference pyramid $\hat{K}(\hat{x}, \hat{y}, \hat{z})$ taken as the unit symmetrical pyramid, centered at the origin (see Fig. 1.1) to a pyramid $K(x, y, z)$ of vertices $S_i = (x_i, y_i, z_i)$ is

\[
F = \frac{1}{4} \left[ (S_1 + S_2 + S_3 + S_4) + \hat{x} (-S_1 + S_2 + S_3 - S_4) + \hat{y} (-S_1 - S_2 + S_3 + S_4) \\
+ \hat{z} (4S_5 - S_1 - S_2 - S_3 - S_4) + \frac{\hat{x}\hat{y}}{1 - \hat{z}} (S_1 + S_3 - S_2 - S_4) \right].
\]  

(1.4)

The finite element space $V_h$ on $\Omega$ is given by

\[
V_h = \left\{ u \in L^2(\Omega) \mid u|_K \in \left( P_F(K) \right)^n \right\},
\]
where $\mathbb{P}_r^F$ is the real space of order $r$ for an element $K$ of the mesh defined by

$$\mathbb{P}_r^F(K) = \left\{ u \mid u \circ F \in \left( \mathbb{P}_r(\hat{K}) \right)^n \right\}.$$  

The finite element space $\hat{P}_r$ of order $r$ on the reference element $\hat{K}$ is

- **Tetrahedron**: $\mathbb{P}_r(\hat{x}, \hat{y}, \hat{z}) = \{ \hat{x}^i \hat{y}^j \hat{z}^k, i + j + k \leq r \}$ of dimension $n_r = \frac{(r+1)(r+2)(r+3)}{6}$, and $\hat{K}$ is the unit tetrahedron $0 \leq \hat{x} + \hat{y} + \hat{z} \leq 1$;

- **Hexahedron**: $\mathbb{Q}_r(\hat{x}, \hat{y}, \hat{z}) = \{ \hat{x}^i \hat{y}^j \hat{z}^k, i, j, k \leq r \}$ of dimension $n_r = (r+1)^3$, and $\hat{K}$ is the unit cube $[0,1]^3$;

- **Wedge**: $\mathbb{P}_r(\hat{x}, \hat{y}, \hat{z}) \otimes \mathbb{P}_r(\hat{z}) = \{ \hat{x}^i \hat{y}^j \hat{z}^k, i + j \leq r, k \leq r \}$ of dimension $n_r = \frac{(r+1)^2(r+2)}{2}$, and $\hat{K}$ is the unit wedge $0 \leq \hat{x} + \hat{y}, \hat{z} \leq 1$;

- **Pyramid**: $\mathbb{P}_r(\hat{x}, \hat{y}, \hat{z}) \oplus \sum_{0 \leq k \leq r-1} \left( \frac{\hat{x} \hat{y}}{1 - \hat{z}} \right)^{r-k} \mathbb{P}_k(\hat{x}, \hat{y})$ of dimension $n_r = \frac{(r+1)(r+2)(2r+3)}{6}$, when $\hat{K}$ is the unit symmetrical pyramid, centered at the origin.

The finite element space is typical for tetrahedra, hexahedra and wedges (Ciarlet [17]) but is less standard in the case of a pyramidal element.

This choice of finite element space is optimal, that is by choosing this finite element space, the final error estimate is in $O(h^{r+1})$ in $L^2$-norm, whereas choosing any subspace included in this one leads to a convergence of at most $O(h^r)$ (see [7]). This can be seen by displaying the dispersion error (see Cohen [11] for the definition of the dispersion analysis, and how to perform it) on a periodic mesh of space step $h$ containing non-affine pyramids, whose cell is presented on Fig. 1.2. As shows Fig. 1.3 displaying the dispersion error obtained for Maxwell’s equations, using the optimal finite element space provides a dispersion error in $O(h^{2r+1})$ (see Pernet [20] for the factor $2r + 1$), whereas the use of $\mathbb{P}_r$ as for tetrahedra leads to a low convergence rate.

![Fig. 1.2. Periodic pattern for the hybrid case, with distorted pyramids (dark) and tetrahedra (light)](image)

**Definition 1.2.** Let $(\varphi_i)_{1 \leq i \leq n_r}$ be a base of $V_h$, let $M_h$ the mass matrix for the pyramid $K$, defined by

$$ (M_h)_{i,j} = \int_K M \varphi_i \cdot \varphi_j \, dx \quad (1.5) $$
the stiffness matrix $R_h$ such that

$$(R_h)_{i,j} = \int_K \sum_{1 \leq k \leq d} \left( A_k \frac{\partial \phi_i}{\partial x_k} \cdot \phi_j - B_k \phi_i \cdot \frac{\partial \phi_j}{\partial x_k} \right) dx$$

and the flux matrix $S_h$ defined by

$$(S_h)_{i,j} = \int_{\partial K} \sum_{1 \leq k \leq d} \left( N_1 \{ \phi_j \} + N_2 [\phi_j] \right) \cdot \phi_i ds.$$ 

The space discretization finally writes as

$$\frac{d}{dt} M_h U - R_h U + S_h U = 0.$$ 

### 1.3. Time Discretization

Using any explicit time scheme, for example the low-storage Runge-Kutta scheme of order 4 (Carpenter and Kennedy [21]), the time discretization writes as

$$U^{n+1} = U^n$$

$$\rho = U^n$$

for $i = 1$ to $5$

$$\rho = \alpha_i \rho + \Delta t(M_h)^{-1}(R_h - S_h)(U^n + \beta_i \rho)$$

end for

For each time-step, we then have to
compute the matrix-vector products $R_h U^n$ and $S_h U^n$;

solve a linear system $(M_h) X = Y$.

We first present an efficient technique to construct $M_h$ and solve the linear system for pyramidal elements.

2. Mass Matrix

2.1. Dense Mass Matrix with Nodal Basis Functions

Using the $H^1$-conforming transformation (Monk [22]) of the basis functions from any element $K$ of the mesh to the reference element

$$\hat{\varphi}_i = \varphi_i \circ F^{-1}, \quad (2.1)$$

the mass matrix writes as

$$(M_h)_{i,j} = \int_K M \varphi_i \varphi_j \, dx \, dy \, dz = \int_{\hat{K}} M |D\varphi| \hat{\varphi}_i \hat{\varphi}_j \, d\hat{x} \, d\hat{y} \, d\hat{z}. \quad (2.2)$$

The nodal basis functions on $\hat{K}$ are obtained by inverting a Vandermonde system as explained in [7]. The resulting mass matrix is dense and has no particular property.

2.2. Sparse Mass Matrix with Orthogonal Basis Functions

2.2.1. Orthogonal Basis Functions

To have a sparser mass matrix, we consider orthogonal basis functions. We denote by $P_{i,j}^m(x)$ the orthonormalized Jacobi polynomial of order $m$, orthogonal for the weight $(1-x)^i(1+x)^j$, and $\xi^G$ the points of Gauss-Legendre on $[0, 1]$ (cf Hammer, Marlowe and Stroud [23]).

Proposition 2.1. The following set of basis functions is an orthogonal base of $\hat{P}_r$

- **Hexahedron**

$$P_{i_1}^{0,0} \left( \frac{2\hat{x}}{1 - \hat{y}} - 1 \right) (1 - \hat{y})^{i_1} P_{i_2}^{2i_1+1,0} \left( 2\hat{y} - 1 \right) P_{i_3}^{2,0} \varphi_i^G(\hat{z}), \quad 0 \leq i_1, i_2, i_3 \leq r,$$

where

$$\varphi_i^G(\hat{x}) = \prod_{j \neq i} \frac{\hat{x} - \xi^G_j}{\xi^G_i - \xi^G_j}.$$

- **Wedge**

$$P_{i_1}^{0,0} \left( \frac{\hat{x}}{1 - \hat{z}} \right) P_{i_2}^{0,0} \left( \frac{\hat{y}}{1 - \hat{z}} \right) (1 - \hat{z})^{max(i_1, i_2)} P_{i_3}^{2max(i_1, i_2)+2,0} \left( 2\hat{z} - 1 \right), \quad 0 \leq i_1, i_2 \leq r, 0 \leq i_3 \leq r - max(i_1, i_2),$$

- **Pyramid**

$$P_{i_1}^{0,0} \left( \frac{\hat{x}}{1 - \hat{z}} \right) P_{i_2}^{0,0} \left( \frac{\hat{y}}{1 - \hat{z}} \right) (1 - \hat{z})^{max(i_1, i_2)} P_{i_3}^{2max(i_1, i_2),0} \left( 2\hat{z} - 1 \right), \quad 0 \leq i_1, i_2 \leq r, 0 \leq i_3 \leq r - max(i_1, i_2),$$
We use the orthogonal base of \(?^2.2.2.\) Fast Algorithm for Pyramidal Elements

To make the computation easier, we write the integrals on the unit cube and the change of variable provides the transformation.

The following set of basis functions is an orthogonal base of \(P_r\).

\[P^{0,0}_i \left( \frac{2\hat{x}}{1 - \hat{y} - \hat{z}} - 1 \right) P^{2i_1+1,0}_{i_2} \left( \frac{2\hat{y}}{1 - \hat{z}} - 1 \right) \left( 1 - \hat{y} - \hat{z} \right)^{i_1} P^{2(i_1+i_2)+2,0}_{i_3} (2\hat{z} - 1) (1 - \hat{z})^{i_2},\]

\(0 \leq i_1 + i_2 + i_3 \leq r.\)

**Proof.** See Proposition 1.10 in [7] for pyramidal elements. The proof is similar for the other types of elements.

When the element is affine, we can use \(\hat{P}_r = P_r\) (see Theorem 1.4 in [7]).

**Proposition 2.2.** The following set of basis functions is an orthogonal base of \(P_r\).

For \(0 \leq i_1 + i_2 + i_3 \leq r,\)

- **Hexahedron**

\[P^{0,0}_i (2\hat{x} - 1) P^{0,0}_{i_2} (2\hat{y} - 1) P^{0,0}_{i_3} (2\hat{z} - 1),\]

- **Wedge**

\[P^{0,0}_i \left( \frac{2\hat{x}}{1 - \hat{y}} - 1 \right) (1 - \hat{y})^{i_1} P^{2i_1+1,0}_{i_2} (2\hat{y} - 1) \varphi_{i_3}^G(\hat{z}),\]

- **Pyramid**

\[P^{0,0}_i \left( \frac{\hat{x}}{1 - \hat{z}} \right) P^{0,0}_{i_2} \left( \frac{\hat{y}}{1 - \hat{z}} \right) (1 - \hat{z})^{i_1+i_2} P^{2(i_1+i_2)+2,0}_{i_3} (2\hat{z} - 1),\]

- **Tetrahedron**

\[P^{0,0}_i \left( \frac{2\hat{x}}{1 - \hat{y} - \hat{z}} - 1 \right) P^{2i_1+1,0}_{i_2} \left( \frac{2\hat{y}}{1 - \hat{z}} - 1 \right) \left( 1 - \hat{y} - \hat{z} \right)^{i_1} P^{2(i_1+i_2)+2,0}_{i_3} (2\hat{z} - 1) (1 - \hat{z})^{i_2}.\]

**Proof.** The proof is similar to the proof of Proposition 2.1.

### 2.2.2. Fast Algorithm for Pyramidal Elements

We use the orthogonal base of \(\hat{P}_r\) defined in Proposition 2.1 as set of basis functions. To make the computation easier, we write the integrals on the unit cube \(\hat{Q}\) by using the following transformation \(T\) from the unit cube \(\hat{Q}\) on the reference pyramid \(\hat{K}\)

\[T : \begin{cases} \hat{x} = (1 - \hat{z})(2\bar{x} - 1) \\ \hat{y} = (1 - \hat{z})(2\bar{y} - 1) \\ \hat{z} = \bar{z}. \end{cases} \tag{2.3}\]

For any function \(f\), we denote

\[\bar{f}(\bar{x}, \bar{y}, \bar{z}) = \hat{f}(\hat{x}, \hat{y}, \hat{z}),\]

and the change of variable provides

\[\int_{\hat{K}} \hat{f}(\hat{x}, \hat{y}, \hat{z}) \, d\hat{x} \, d\hat{y} \, d\hat{z} = \int_{\hat{Q}} 4 \, \bar{f}(\bar{x}, \bar{y}, \bar{z}) (1 - \bar{z})^2 \, d\bar{x} \, d\bar{y} \, d\bar{z}. \tag{2.4}\]

The mass matrix written on the unit cube \(\hat{Q}\) is then

\[(M_h)_{ij} = 4 \int_{\hat{Q}} M \left| \overline{DF} \right| \varphi_i \varphi_j (1 - \bar{z})^2 \, d\bar{x} \, d\bar{y} \, d\bar{z},\]
where the jacobian $|DF|$ of $F$ on $\bar{Q}$ can be written as (see Lemma 3.5 in [7])

$$|DF| = A + B_1(2\bar{x} - 1) + B_2(2\bar{y} - 1) + C(2\bar{x} - 1)(2\bar{y} - 1).$$

We recall the following property of Jacobi polynomial (Szegő [24])

**Property 2.1.**

$$tP^{i,j}_k(t) = \gamma^{i,j}_k P^{i,j}_k(t) + \alpha^{i,j}_k P^{i,j}_k(t) + \beta^{i,j}_k P^{i,j}_{k-1}(t),$$

where

$$\alpha^{i,j}_k = -a_{i,j,k} \frac{(2k + i + j)(i^2 - j^2)}{2k + i + j}, \quad \beta^{i,j}_k = b_{i,j,k} \frac{2(k + i)(k + j)(2k + i + j + 2)}{2k + i + j}, \quad \gamma^{i,j}_k = c_{i,j,k} 2(k + 1)(k + i + j + 1)$$

and $a_{i,j,k}, b_{i,j,k}$ and $c_{i,j,k}$ are coefficients due to orthonormalisation of Jacobi polynomial $P^{i,j}_k$.

Using Property 2.1, we decompose the mass matrix for all $i = (i_1, i_2, i_3), 1 \leq i \leq n_r$ and for all $j_3, 0 \leq j_3 \leq r$

- **Term in $A$:**

$$\int_{\bar{Q}} \bar{\varphi}_i \bar{\varphi}_j (1 - \bar{z})^2 \, d\bar{x} \, d\bar{y} \, d\bar{z} =$$

$$\int_0^1 \frac{1}{\delta_{j_1j_1}} P_{i_1}^{0,0}(2\bar{x} - 1) P_{j_1}^{0,0}(2\bar{x} - 1) \, d\bar{x}$$

$$\int_0^1 \frac{1}{\delta_{j_2j_2}} P_{i_2}^{0,0}(2\bar{y} - 1) P_{j_2}^{0,0}(2\bar{y} - 1) \, d\bar{y}$$

$$\int_0^1 (1 - \bar{z})^{\max(i_1,i_2)+\max(j_1,j_2)+2} P_{i_3}^{2\max(i_1,i_2)+2,0}(2\bar{z} - 1) P_{j_3}^{2\max(j_1,j_2)+2,0}(2\bar{z} - 1) \, d\bar{z};$$

- **Term in $B_1$:**

$$\int_{\bar{Q}} \bar{\varphi}_i \bar{\varphi}_j (2\bar{x} - 1)(1 - \bar{z})^2 \, d\bar{x} \, d\bar{y} \, d\bar{z} =$$

$$\int_0^1 \frac{1}{\gamma_{i_1j_1}^0,\delta_{i_1j_1}^0 + \beta_{i_1j_1}^0} (2\bar{x} - 1) P_{i_1}^{0,0}(2\bar{x} - 1) P_{j_1}^{0,0}(2\bar{x} - 1) \, d\bar{x}$$

$$\int_0^1 \frac{1}{\delta_{j_2j_2}} P_{i_2}^{0,0}(2\bar{y} - 1) P_{j_2}^{0,0}(2\bar{y} - 1) \, d\bar{y}$$

$$\int_0^1 (1 - \bar{z})^{\max(i_1,i_2)+\max(j_1,j_2)+2} P_{i_3}^{2\max(i_1,i_2)+2,0}(2\bar{z} - 1) P_{j_3}^{2\max(j_1,j_2)+2,0}(2\bar{z} - 1) \, d\bar{z};$$

- **Terms in $B_2$ and $C$:** treated in a similar manner to $B_1$
The mass matrix is finally computed as follows

- \( M_h[i, i] = A \)
- \( M_h[i, (i_1 + 1, i_2, j_3)] = B_1 \gamma_{i_1} \epsilon_{i_1, i_1 + 1}(i_3, j_3) \)
- \( M_h[i, (i_1 - 1, i_2, j_3)] = B_1 \beta_{i_1} \epsilon_{i_1, i_1 - 1}(i_3, j_3) \)
- \( M_h[i, (i_1, i_2 + 1, j_3)] = B_2 \epsilon_{i_2, i_2 + 1}(i_3, j_3) \)
- \( M_h[i, (i_1, i_2 - 1, j_3)] = B_2 \beta_{i_2} \epsilon_{i_2, i_2 - 1}(i_3, j_3) \)
- \( M_h[i, (i_1 + 1, i_2 + 1, j_3)] = C_\gamma \epsilon_{i_1, i_1 + 1}(i_3, j_3) \gamma_{i_2} \epsilon_{i_2, i_2 + 1}(i_3, j_3) \)
- \( M_h[i, (i_1 + 1, i_2, j_3)] = C_\gamma \epsilon_{i_1, i_1 + 1}(i_3, j_3) \beta_{i_2} \epsilon_{i_2, i_2 - 1}(i_3, j_3) \)
- \( M_h[i, (i_1, i_2 + 1, j_3)] = C_\beta \epsilon_{i_2, i_2 + 1}(i_3, j_3) \gamma_{i_2} \epsilon_{i_2, i_2 + 1}(i_3, j_3) \)
- \( M_h[i, (i_1, i_2 - 1, j_3)] = C_\beta \epsilon_{i_2, i_2 - 1}(i_3, j_3) \beta_{i_2} \epsilon_{i_2, i_2 - 1}(i_3, j_3) \)

Using orthogonal basis functions, the mass matrix contains \( O(r^4) \) non-zero entries instead of \( O(r^6) \), which will be useful for the matrix-vector product. Integrals \( \epsilon_{i_1, j_1}(i_3, j_3) \) are precomputed, so that the cost of computation is finally in \( O(r^4) \).

### 2.2.3. Mass Matrix for the Other Elements

For hexahedral elements, the mass matrix is diagonal thanks to the mass-lumping. For tetrahedral elements, as the jacobian is constant, the mass matrix is also diagonal when using orthogonal basis functions.

For wedge elements, as we have tensorisation and mass-lumping in \( z \), the mass matrix is already block diagonal on each element, even for nodal elements for which the number of non-zero entries is in \( O(r^5) \). Using orthogonal basis functions make each block sparser, but the gain is far less spectacular than for pyramidal elements. The same algorithm as for pyramidal elements could be used for wedge elements to speed-up the computation, but is not presented here.

### 2.3. Diagonal Mass Matrix with Warburton’s Trick

In the discontinuous case, the resolution of the linear system can be avoided by using the following non-\( H^1 \)-conforming transformation introduced by Warburton [14]

\[
\hat{\varphi}_i = \frac{1}{\sqrt{|DF|}} \varphi_i \circ F^{-1}.
\]  

Using this transformation on the reference element, the mass matrix writes as

\[
(M_h)_{i,j} = \int_K \varphi_i \cdot \varphi_j dx = \int_K |DF| \frac{\hat{\varphi}_i}{\sqrt{|DF|}} \cdot \frac{\hat{\varphi}_j}{\sqrt{|DF|}} dx = \int_K \hat{\varphi}_i \cdot \hat{\varphi}_j dx,
\]

that is the mass matrix is independent of the geometry. If orthonormal basis functions are used, the matrix is equal to the identity.
This method was originally intended by Warburton to handle efficiently curvilinear tetrahedra, when the jacobian $|DF|$ tends to a constant. In that case, the method performs very well. Here we apply this technique beyond its area of application. As this transformation obviously does not preserve the constants, the method will probably not be consistent in this case.

3. Fast Matrix-Vector Product

3.1. Introduction

In the time discretization (see algorithm 1.9), a needed step consists of performing the following matrix-vector product

$$y^n = (R_h - S_h) U^n$$

Storing the matrix $R_h - S_h$ as a sparse matrix and performing a standard matrix-vector product would be an expensive solution as the memory required to store such a large matrix may become too important, especially when a high order is used.

Since inside each element (except hexahedron), all the degrees of freedom are interacting with each other, i.e.

$$\int_K \frac{\partial \varphi_k}{\partial x_i} \varphi_j \, dx \neq 0, \quad \forall j, k$$

the number of non-zero entries in matrix $R_h - S_h$ is equal to $O(n_e r^6)$ where $n_e$ is the number of elements in the mesh and $r$ the order of approximation. The computational time required for the matrix-vector product would therefore be in $O(n_e r^6)$ too if the matrix $R_h - S_h$ is stored.

Another solution, well-known for tetrahedra (Hesthaven [1]) consists in performing the matrix-vector product without storing the matrix. For nodal basis functions, the computational time will remain in $O(n_e r^6)$, whereas for orthogonal functions, the tensorization of basis functions induce fast algorithms in $O(n_e r^4)$ as pointed out in Warburton’s thesis [25].

In the following section, we detail an efficient algorithm for the discontinuous Galerkin formulation used in this paper.

3.2. $H^1$-conforming Transformation

3.2.1. General Method

We are considering the following matrix vector product

$$y_j = \int_K \sum_{1 \leq i \leq d} \left( A_i U \cdot \frac{\partial \varphi_j}{\partial x_i} - B_i \frac{\partial U}{\partial x_i} \cdot \varphi_j \right) \, dx - \int_{\partial K} (N_1 \{U\} + N_2 [U]) \cdot \varphi_j \, ds$$

We use $F^{-1}$ to transform any element $K$ of the mesh into the reference element $\hat{K}$, and $T$ defined by equation 2.3 to transform the reference pyramid $\hat{K}$ into the unit cube $\tilde{Q}$. Let us denote by $G = F \circ T$, the transformation from the pyramid $K$ into the unit cube $\tilde{Q}$. We also use a transformation $g^{-1}$ from a face $\partial K$ to a reference face $\partial \tilde{Q}$. As

$$\tilde{u} = \sum_k \bar{u}_k \tilde{\varphi}_k,$$
$y_j$ finally writes as

$$y_j = \int_{\tilde{Q}} |\tilde{D}\tilde{G}| \sum_{1 \leq i \leq d} \sum_{1 \leq l \leq d} \left( A_i \tilde{u} \cdot (\tilde{D}\tilde{G}^{-1})_{l,i} \frac{\partial \tilde{\varphi}_j}{\partial \tilde{x}_l} - B_i (\tilde{D}\tilde{G}^{-1})_{l,i} \frac{\partial \tilde{u}}{\partial \tilde{x}_l} \cdot \tilde{\varphi}_j \right) d\tilde{x} d\tilde{y} d\tilde{z}$$

$$\quad - \int_{\partial \tilde{Q}} |\tilde{D}g| (N_1 \{\tilde{u}\} + N_2 [\tilde{u}]) \cdot \tilde{\varphi}_j d\tilde{s}.$$

Volume integrals are evaluated with quadrature formulas $(\omega_m, \xi_m)$, suited for the cube $\tilde{Q}$, whereas surface integrals are evaluated with quadrature formulas $(\omega'_n, \xi'_n)$ suited for the faces of cube $\partial \tilde{Q}$. Let us define

$$\bar{U} = -(N_1 \{\tilde{u}(\xi'_n)\} + N_2 [\tilde{u}(\xi'_n)]).$$

We finally write

$$y_j = \sum_m \omega_m |\tilde{D}\tilde{G}|(\xi_m) \sum_{1 \leq i \leq d} \sum_{1 \leq l \leq d} \left( (\tilde{D}\tilde{G}^{-1})_{l,i} A_i \tilde{u} \cdot \frac{\partial \tilde{\varphi}_j}{\partial \tilde{x}_l} - B_i (\tilde{D}\tilde{G}^{-1})_{l,i} \frac{\partial \tilde{u}}{\partial \tilde{x}_l} \cdot \tilde{\varphi}_j \right)(\xi_m)$$

$$\quad + \sum_n \omega'_n |\tilde{D}g|(\xi'_n) \bar{U} \cdot \tilde{\varphi}_j(\xi'_n).$$

We decompose this matrix-vector into several stages.

**For volume integrals**:

1. Evaluate

$$v_m = \tilde{u}(\xi_m) = \sum_k \tilde{u}_k \tilde{\varphi}_k(\xi_m), \quad dv_m^l = \frac{\partial \tilde{u}}{\partial \tilde{x}_l}(\xi_m) = \sum_k \tilde{u}_k \frac{\partial \tilde{\varphi}_k}{\partial \tilde{x}_l}(\xi_m)$$

2. Apply geometry and physics coefficients

$$v_m^{1,l} = \sum_{1 \leq i \leq d} \omega_m |\tilde{D}\tilde{G}|(\xi_m) D\tilde{G}^{-1}_{l,i} A_i v_m, \quad v_m^2 = -\sum_{1 \leq i,l \leq d} \omega_m |\tilde{D}\tilde{G}|(\xi_m) B_i \tilde{D}\tilde{G}^{-1}_{l,i} dv_m^l$$

3. Evaluate

$$w_j^1 = \sum_m \sum_{1 \leq l \leq d} v_m^{1,l} \cdot \frac{\partial \tilde{\varphi}_j}{\partial \tilde{x}_l}(\xi_m), \quad w_j^2 = \sum_m v_m^2 \cdot \tilde{\varphi}_j(\xi_m)$$

**For surface integral**:

1. Evaluate

$$s_n = \sum_k u_k \tilde{\varphi}_k(\xi'_n)$$

2. Apply geometry and physics coefficients to get

$$\bar{s}_n = -(N_1 \{s_n\} + N_2 [s_n])$$

$$s_n^1 = \omega'_n |\tilde{D}g|(\xi'_n) \bar{s}_n$$
3. Evaluate

$$w_j^3 = \sum_n s_n^1 \cdot \tilde{\varphi}_j(\xi_n')$$

Get the result vector:

$$y_j = w_j^1 + w_j^2 + w_j^3$$

We consider tensorized quadrature points

$$\xi_m = (\xi_{m_1}, \xi_{m_2}, \xi_{m_3})$$

and tensorized basis functions

$$\varphi_j(x, y, z) = \tilde{\varphi}_{j_1}(\tilde{x}) \tilde{\varphi}_{j_2}(\tilde{y}) \tilde{\varphi}_{j_3}(\tilde{z})$$

We now detail how the different stages containing sums can be decomposed because of tensorization. By performing computations along each coordinate $\tilde{x}$, $\tilde{y}$, $\tilde{z}$, we are able to have sums with only $r + 1$ terms, instead of $(r + 1)^3$ terms if basis functions and quadrature points do not exhibit any tensorization.

3.2.2. Computation of Volume Integrals

1. For $m = (m_1, m_2, m_3)$, we want to evaluate

$$v_m = \sum_{k_1, k_2, k_3} \tilde{\varphi}_{k_1}(\xi_{m_1}) \tilde{\varphi}_{k_2}(\xi_{m_2}) \tilde{\varphi}_{k_3}(\xi_{m_3}) u_{k_1, k_2, k_3}.$$

We notice $\tilde{\varphi}_{j_1, j_2}$ when the basis function depends on $j_1$ and $j_2$. This triple sum is split into three single sums

$$u_{k_1, k_2, m_3}^1 = \sum_{k_3} \tilde{\varphi}_{k_3}(\xi_{m_3}) u_{k_1, k_2, k_3}$$

$$u_{k_1, m_2, m_3}^2 = \sum_{k_2} \tilde{\varphi}_{k_2}(\xi_{m_2}) u_{k_1, k_2, m_3}$$

$$v_{m_1, m_2, m_3} = \sum_{k_1} \tilde{\varphi}_{k_1}(\xi_{m_1}) u_{k_1, m_2, m_3}^2$$

Remark 3.1. We observe that the dependency has to be “opposite” between $\varphi_j$ and $\xi_m$, otherwise the tensorized structure of quadrature points and the basis functions can not be exploited to get fast algorithms. Semi-tensorized quadrature point could also be considered

$$\xi_m = (\xi_{m_1}, \xi_{m_2}, \xi_{m_3}).$$

The three sums involve only $O(r)$ terms and are computed for $O(r^3)$ values, leading to a cost in $O(r^4)$. Each of these sums can be interpreted as a matrix vector product, i.e.

$$U^1 = C_1 U, \quad U^2 = C_2 U^1, \quad V = C_3 U^2$$

We have found a factorization of matrix $C$

$$C_{m,k} = \tilde{\varphi}_k(\xi_m)$$
which is

\[ C = C_3 C_2 C_1 \]

While matrix \( C \) is dense, the matrices \( C_3, C_2 \) and \( C_1 \) are sparse. Matrices \( C_3, C_2, C_1 \) are independent from the geometry, and are precomputed once for each reference element. Hence, we have

\[ V = C U \]

For \( n = n_1, n_2, n_3 \), we now want to evaluate

\[ dV_n^l(\xi_n) = \sum_k \tilde{u}_k \frac{\partial \tilde{\varphi}_k}{\partial x_l}(\xi_n) \]

Since for all elements, we have the inclusion \( \tilde{V}_r \subset Q_r \), we can write

\[ \tilde{u}(x, y, z) = \sum_k \tilde{u}_k \tilde{\varphi}_k(x, y, z) = \sum_{m_1, m_2, m_3} v_{m_1, m_2, m_3} \psi_{m_1}(x) \psi_{m_2}(y) \psi_{m_3}(z) \]

where \( \psi_{m_1}, \psi_{m_2}, \psi_{m_3} \) are Lagrange interpolation basis function associated respectively with points \( \xi_{m_1}, \xi_{m_2} \) and \( \xi_{m_3} \)

\[ \psi_{m_1}(x) = \prod_{n_1 \neq m_1} \frac{x - \xi_{n_1}}{\xi_{m_1} - \xi_{n_1}} \quad \psi_{m_2}(y) = \prod_{n_2 \neq m_2} \frac{y - \xi_{n_2}}{\xi_{m_2} - \xi_{n_2}} \quad \psi_{m_3}(z) = \prod_{n_3 \neq m_3} \frac{z - \xi_{n_3}}{\xi_{m_3} - \xi_{n_3}} \]

We have

\[ \nabla \psi_{m_1}(x) \psi_{m_2}(y) \psi_{m_3}(z) = \left| \begin{array}{c}
\sum_{m_1, m_2, m_3} v_{m_1, m_2, m_3} \frac{d\psi_{m_1}}{dx}(x) \psi_{m_2}(y) \psi_{m_3}(z) \\
\sum_{m_1, m_2, m_3} v_{m_1, m_2, m_3} \psi_{m_1}(x) \frac{d\psi_{m_2}}{dy}(y) \psi_{m_3}(z) \\
\sum_{m_1, m_2, m_3} v_{m_1, m_2, m_3} \psi_{m_1}(x) \psi_{m_2}(y) \frac{d\psi_{m_3}}{dz}(z)
\end{array} \right| \]

Since we have

\[ \psi_{m_1}(\xi^{n_1}) = \delta_{m_1, n_1}, \quad \psi_{m_2}(\xi^{n_2}) = \delta_{m_2, n_2}, \quad \psi_{m_3}(\xi^{n_3}) = \delta_{m_3, n_3} \]

Triple sums over \( m_1, m_2, m_3 \) are reduced to single sums

\[ \left( \frac{\partial v}{\partial x} \right)_{n_1, n_2, n_3} = \sum_{m_1} \frac{d\psi_{m_1}}{dx}(\xi_{n_1}) v_{m_1, n_2, n_3} \]

\[ \left( \frac{\partial v}{\partial y} \right)_{n_1, n_2, n_3} = \sum_{m_2} \frac{d\psi_{m_2}}{dy}(\xi_{n_2}) v_{m_1, n_2, n_3} \]

\[ \left( \frac{\partial v}{\partial z} \right)_{n_1, n_2, n_3} = \sum_{m_3} \frac{d\psi_{m_3}}{dz}(\xi_{n_3}) v_{m_1, n_2, n_3} \]

These operations can be viewed as matrix vector product

\[ dV = RV \]
where the matrix $R$ is sparse and independent from the geometry, and can be precomputed once for each reference element.

Eventually, we have exhibited the following factorization

$$dV = R C U.$$ 

2. We compute

$$v^1_m = \sum_{1 \leq i \leq d} \omega_m \tilde{D}G_l(\xi_m) \tilde{D}G_{l,i}^{-1} A_i v_m, \quad v^2_m = \sum_{1 \leq i \leq d} \omega_m \tilde{D}G_l(\xi_m) B_i \left( \sum_{1 \leq l \leq d} \tilde{D}G_{l,i}^{-1} d v^l_m \right)$$

The complexity of this operation is in $O(r^3)$, and can be viewed as a matrix-vector product

$$V^1 = AV, \quad V^2 = B dV$$

where matrices $A$ and $B$ are block-diagonal, each block being related to a quadrature point, and depend on the geometry.

3. We are interested in the stage

$$w^1_j = \sum_{m,l} v^1_m \cdot \frac{\partial \tilde{\varphi}_j}{\partial x_l}(\xi_m)$$

which is the transpose operation to the computation of derivative of basis functions on quadrature points. Therefore, we have

$$W^1 = C^* R^* V^1$$

We are now interested in the step

$$w^2_j = \sum_m v^2_m \cdot \tilde{\varphi}_j(\xi_m)$$

which can be interpreted as

$$W^2 = C^* V^2$$

that is

$$W^2 = C^*_1 C^*_2 C^*_3 V^2$$

**Remark 3.2.** Using the basis functions of Equation 2.1, we have $C = Id$, which makes computations faster for hexahedra. That is why we want to have high percentage of hexahedra in a mesh.

### 3.2.3. Computation of Surface Integrals

1. We are interested in the stage

$$s_p = \tilde{u}(\xi'_p) = \sum_k \tilde{u}_k \tilde{\varphi}_k(\xi'_p)$$

Since we are considering faces of the cube, we have three families of quadrature points

$$(\delta, \xi'_{p_2}, \xi'_{p_3}), \quad (\xi'_{p_1}, \delta, \xi'_{p_3}), \quad (\xi'_{p_1}, \xi'_{p_2}, \delta)$$
where $\delta$ is equal to 0 or 1. The starting point is to consider the expansion with basis functions of the cube, ie

$$\tilde{u}(\tilde{x}, \tilde{y}, \tilde{z}) = \sum_{m_1, m_2, m_3} v_{m_1, m_2, m_3} \psi_{m_1}(\tilde{x}) \psi_{m_2}(\tilde{y}) \psi_{m_3}(\tilde{z})$$

Then we evaluate $u$ on the families of points

$$u^1_{m_2, m_3} = \tilde{u}(\delta, \xi_{m_2}, \xi_{m_3})$$
$$u^2_{m_1, m_3} = \tilde{u}(\xi_{m_1}, \delta, \xi_{m_3})$$
$$u^3_{m_1, m_2} = \tilde{u}(\xi_{m_1}, \xi_{m_2}, \delta)$$

These operations consist of single sums

$$u^1_{m_2, m_3} = \sum_{m_1} \psi_{m_1}(\delta) v_{m_1, m_2, m_3}$$
$$u^2_{m_1, m_3} = \sum_{m_2} \psi_{m_2}(\delta) v_{m_1, m_2, m_3}$$
$$u^3_{m_1, m_2} = \sum_{m_3} \psi_{m_3}(\delta) v_{m_1, m_2, m_3}$$

which can be interpreted as matrix vector products with sparse matrices $P_1, P_2, P_3$

$$U^1 = P_1 V, \quad U^2 = P_2 V, \quad U^3 = P_3 V$$

Then, if the first face is a quadrangular face, we can split the computation

$$s^1_{p_2, p_3} = \sum_{m_2, m_3} \psi_{m_2}(\xi'_{p_2}) \psi_{m_3}(\xi'_{p_3}) u^1_{m_2, m_3}$$

into two stages

$$z_{m_2, p_3} = \sum_{m_3} \psi_{m_3}(\xi'_{p_3}) u^1_{m_2, m_3}$$
$$s^1_{p_2, p_3} = \sum_{m_2} \psi_{m_2}(\xi'_{p_2}) z_{m_2, p_3}$$

which can be interpreted as matrix vector products with sparse matrices

$$S^1 = T_2 T_1 U^1$$

If the first face is a triangle, since we are using symmetric quadrature points for triangles [26] which are not tensorized, we only have

$$S^1 = T U^1$$

where matrix $T$ is dense, but only restricted to the face. Therefore the complexity of computation of $s_p$ is in $O(r^3)$ if the element is an hexahedron (only quadrangular faces) and in $O(r^4)$ for other elements, because of the presence of triangular faces. Of course, for other elements, some faces of the cube are not treated since they are reduced to a single point in the real element $K$. For pyramids, the face $z = 1$ is not treated.

2. We compute

$$\bar{s}_n = -(N_1 \{s_n\} + N_2 [s_n])$$
$$s^1_n = \omega'_n |Dg| (\xi'_{p_n}) \bar{s}_n$$
3. We are interested in the stage:

\[ w^3_j = \sum_n s_n^1 \cdot \tilde{\varphi}_j (\xi'_n) \]

This stage is exactly the transpose operation to the computation of \( s_n \), therefore the computation of \( w^3 \) is done by using transpose of matrices defined in the previous subsubsection (matrices \( P_1, P_2, P_3, T_1, T_2 \)).

3.3. With Warburton’s Trick

Using the transformation 2.6, the stiffness matrix is transformed into

\[
(R_h)_{j,k} = \int_{\Omega} |DF| \sum_{1 \leq i \leq d} A_i \frac{\partial (\tilde{\varphi}_k)}{\partial x_i} \cdot \tilde{\varphi}_j d\hat{x} - \int_{\Omega} |DF| \sum_{1 \leq i \leq d} B_i \frac{\partial \tilde{\varphi}_k}{\partial x_i} d\hat{x} \\
= \int_{\Omega} \sum_{1 \leq i \leq d} A_i \frac{\partial \tilde{\varphi}_k}{\partial x_i} \cdot \tilde{\varphi}_j d\hat{x} - \int_{\Omega} \sum_{1 \leq i \leq d} B_i \frac{\partial \tilde{\varphi}_k}{\partial x_i} d\hat{x} \\
+ \frac{1}{2} \int_{\Omega} \sum_{1 \leq i \leq d} (B_i - A_i) \frac{1}{|DF|} \frac{\partial |DF|}{\partial x_i} [\tilde{\varphi}_k \cdot \tilde{\varphi}_j] d\hat{x}
\]

We can use the fast matrix vector product detailed above with this modified stiffness matrix without important increase in cost. Moreover, since the mass matrix is equal to identity, this leads to faster computations as shows Table 4.8. However, the stiffness matrix involves the derivative of the jacobian and this has to be treated carefully with pyramids. Indeed, for non-affine pyramids, the derivative of jacobian is singular :

\[
\frac{\partial |DF|}{\partial \hat{x}} = B_1 \frac{1}{1 - \hat{z}} + C \frac{\hat{y}}{(1 - \hat{z})^2} = B_1 \frac{1}{1 - \hat{z}} + C \frac{2\hat{y} - 1}{1 - \hat{z}}
\]

It is preferable, for the volume integrals, to use a Gauss-Jacobi quadrature formula exact for polynomials in \((1 - \hat{z}) Q_r\) instead of \((1 - \hat{z})^2 Q_r\) as chosen classically for pyramids.

4. Comparison of Several Types of Elements

Experiments to compare the elements previously constructed are conducted on Maxwell’s equations for which \( d = 3 \) and \( n_s = 6 \), and where \( u^m = E_x, E_y, E_z, H_x, H_y \) or \( H_z \). The system is the following for \( E \) and \( H \) in \( \mathbb{R}^3 \)

\[
\begin{cases}
\frac{dE}{dt} - \text{rot} H = 0 \\
\frac{dH}{dt} + \text{rot} E = 0
\end{cases} \quad (4.1)
\]

where the matrices \( M \) and \( A_i \) are easily deduced from Equation 4.1, and the matrices \( B_i \) are obtained by taking the transpose of \( A_i \).

We consider a cubic cavity \([-5, 5]^3\) meshed with different types of elements, depending on the studied case, with a gaussian source

\[ f = x e^{-\alpha r^2} e_x \]
4.1. Hexahedron

The orthogonal base of Proposition 2.1 provides a diagonal mass matrix for non-affine hexahedra, while the tensor product of Legendre polynomials proposed by Kirby et al. [9] (with \( P_1 = P_2 = P_3 = r \)) and Warburton [25] only provides mass-lumping for affine elements.

When the hexahedron is affine, the use of Legendre polynomials combined with a quadrature-free implementation of the matrix-vector product conduces to very efficient computations. In the Table 4.1, we have compared the computation time for one million degrees of freedom for \( Q_r \) and \( P_r \). We observe that for \( P_r \), the cost remains constant for higher orders than for \( Q_r \).

Table 4.1: Computational time spent for 100 iterations of leap-frog scheme for meshes of hexahedra containing 1 million DOFs for unknown \( E_x \).

<table>
<thead>
<tr>
<th>Order</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Q_r )</td>
<td>136s</td>
<td>111s</td>
<td>107s</td>
<td>100s</td>
<td>101s</td>
<td>99s</td>
<td>100s</td>
<td>105s</td>
<td>113s</td>
<td>140s</td>
</tr>
<tr>
<td>( P_r )</td>
<td>214s</td>
<td>157s</td>
<td>135s</td>
<td>117s</td>
<td>111s</td>
<td>105s</td>
<td>106s</td>
<td>101s</td>
<td>97s</td>
<td>99s</td>
</tr>
</tbody>
</table>

For a sharper comparison, we study the \( L^2 \)-error obtained for a cubic cavity meshed with small cubes. Because of symmetry, only the cube \([0, 5\lambda]^3\) is meshed. In Table 4.2, we measure the minimal computational time and minimal number of degrees of freedom necessary to obtain an error less than 1%, for \( P_r \) and \( Q_r \).

Table 4.2: Time step, number of DOFs and computational time necessary to reach an error level below one percent for a cubic cavity (affine mesh) with hexahedra. Use of \( P_r \) with Legendre orthogonal expansion (top) - Use of \( Q_r \) with Lagrange interpolation functions base on Gauss-Legendre points (bottom).

<table>
<thead>
<tr>
<th>Order</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>14</th>
<th>18</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Delta t )</td>
<td>0.035</td>
<td>0.0405</td>
<td>0.0394</td>
<td>0.0395</td>
<td>0.0348</td>
<td>0.0325</td>
<td>0.0272</td>
<td>0.0235</td>
<td>0.0179</td>
</tr>
<tr>
<td>DOFs</td>
<td>540 000</td>
<td>240 065</td>
<td>153 664</td>
<td>111 804</td>
<td>84 480</td>
<td>61 776</td>
<td>43 520</td>
<td>35 910</td>
<td>29 232</td>
</tr>
<tr>
<td>CPU Time</td>
<td>11 818s</td>
<td>2865s</td>
<td>1387s</td>
<td>856s</td>
<td>695s</td>
<td>500s</td>
<td>444s</td>
<td>376s</td>
<td>620s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Order</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>11</th>
<th>16</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Delta t )</td>
<td>0.032</td>
<td>0.036</td>
<td>0.0353</td>
<td>0.0344</td>
<td>0.0381</td>
<td>0.0315</td>
<td>0.0301</td>
<td>0.023</td>
<td>0.0156</td>
<td></td>
</tr>
<tr>
<td>DOFs</td>
<td>512 000</td>
<td>216 000</td>
<td>157 464</td>
<td>117 649</td>
<td>64 000</td>
<td>64 000</td>
<td>46 656</td>
<td>39 304</td>
<td>29 791</td>
<td></td>
</tr>
<tr>
<td>CPU Time</td>
<td>5 575s</td>
<td>1800s</td>
<td>1180s</td>
<td>1013s</td>
<td>417s</td>
<td>580s</td>
<td>426s</td>
<td>1134s</td>
<td>3520s</td>
<td></td>
</tr>
</tbody>
</table>

It seems better to use \( P_r \) than \( Q_r \), for higher orders, but all in all, the two methods give similar computational times. We think that using \( Q_r \) is a better choice than using \( P_r \) since the method also works for non-affine hexahedron, which means implementing a single method for all the cases.

4.2. Wedges

The orthogonal basis functions of Proposition 2.1 are almost the same as Kirby et al. [9] and Warburton [25], replacing Legendre polynomials in \( z \) by Lagrange interpolation functions...
with Gauss points. For nodal functions, the tensorization in $z$ leads to a matrix-vector product in $O(r^3)$ instead of $O(r^4)$ for orthogonal basis. In Table 4.3, the computational times given by both basis are very close, orthogonal functions provide faster computations from order 5.

Table 4.3: Time spent for 100 iterations of leap-frog scheme for meshes of non-affine wedges containing one million DOFs for $E_x$.

<table>
<thead>
<tr>
<th>Order</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodal</td>
<td>296</td>
<td>312</td>
<td>239</td>
<td>342</td>
<td>343</td>
<td>353</td>
<td>400</td>
<td></td>
</tr>
<tr>
<td>Orthogonal</td>
<td>338</td>
<td>326</td>
<td>280</td>
<td>315</td>
<td>292</td>
<td>317</td>
<td>361</td>
<td></td>
</tr>
</tbody>
</table>

Similar experiments than for affine-hexahedron have been conducted to compare the use of $P_r$ or the optimal finite element space for wedges. For $P_r$, we take the orthogonal basis functions of Proposition 2.2. The required number of degrees of freedom and computational time for the experiments can be read in Table 4.4.

Table 4.4: Time step, number of DOFs and computational time necessary to reach an error level below one percent for a cubic cavity meshed with affine prisms. Use of $P_r$ (top) and optimal finite element space (bottom) with orthogonal expansion.

<table>
<thead>
<tr>
<th>Order</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta t$</td>
<td>0.0268</td>
<td>0.0296</td>
<td>0.0309</td>
<td>0.029</td>
<td>0.0285</td>
<td>0.0268</td>
<td>0.0258</td>
<td></td>
</tr>
<tr>
<td>DOFs</td>
<td>351 520</td>
<td>172 955</td>
<td>96 768</td>
<td>84 000</td>
<td>61 440</td>
<td>56 595</td>
<td>47 520</td>
<td></td>
</tr>
<tr>
<td>CPU Time</td>
<td>30 330s</td>
<td>12 920s</td>
<td>7 145s</td>
<td>7 028s</td>
<td>5 560s</td>
<td>6 306s</td>
<td>6 602s</td>
<td></td>
</tr>
</tbody>
</table>

In this case, we do not have the same advantage than for hexahedra where quadrature-free algorithms lead to faster computations. The use of optimal finite element space is then much better, even though the number of degrees of freedom is lower for $P_r$.

4.3. Pyramids

Computations are done for a mesh containing only non-affine pyramids and one million degrees of freedom for unknown $E_x$. In Table 4.5, we compare the CPU time spent for the computation and Cholesky factorization of mass matrix for nodal and orthogonal basis functions, using the algorithm previously described. The computational time needed for nodal functions grows quickly, making the use of orthogonal functions very attractive when $r$ is high. However, the impressive computational time (only spent during the computation of mass matrix) observed for nodal basis is not an important burden, since this operation is performed only once.

In Table 4.6, we display the ratio between the amount of memory used for a dense and a sparse mass matrix with orthogonal basis. The induced reduction of memory requirements is quite interesting for both the matrix and its Cholesky factorisation.
Table 4.5: Time to compute and invert mass matrix with the fast algorithm for pyramid elements

<table>
<thead>
<tr>
<th>Order</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense Matrix</td>
<td>1.91s</td>
<td>7.98s</td>
<td>26.2s</td>
<td>73.6s</td>
<td>177s</td>
<td>391s</td>
<td>786s</td>
</tr>
<tr>
<td>Sparse Matrix</td>
<td>0.577s</td>
<td>0.98s</td>
<td>1.32s</td>
<td>2.27s</td>
<td>4.08s</td>
<td>5.17s</td>
<td>7.62s</td>
</tr>
</tbody>
</table>

Table 4.6: Gain in storage for the matrix and its Cholesky factorization by using orthogonal functions instead of nodal functions for pyramidal elements.

<table>
<thead>
<tr>
<th>Order</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gain for matrix</td>
<td>1.81</td>
<td>2.94</td>
<td>4.46</td>
<td>6.38</td>
<td>8.7</td>
<td>11.4</td>
<td>14.6</td>
</tr>
<tr>
<td>Gain for factorization</td>
<td>1.52</td>
<td>1.91</td>
<td>2.39</td>
<td>2.83</td>
<td>3.38</td>
<td>3.85</td>
<td>4.8</td>
</tr>
</tbody>
</table>

We finally perform a comparison between different solvers to invert the mass matrix: to be exhaustive, we consider a dense, a sparse and an iterative solver using fast matrix vector product deduced from the expression for the mass matrix given in Section 2.1. Table 4.7 shows that the sparse solver outperforms the other solvers. The main drawback of iterative solver is the relatively high number of iterations, since 10 iterations are needed to get a residual below $10^{-12}$.

We compare the computation time spent for 100 leap-frog iterations with a mesh made of only non-affine pyramids only with one million DOFs in Table 4.8. The efficiency of the matrix-vector product seems quite interesting, since with Warburton’s trick, the cost remains almost constant (between 400 and 500s here) with respect to the order of approximation. When a sparse solver is used, the additional increase in cost is quite acceptable if a high accuracy is sought. The cost of the matrix vector product using nodal functions becomes quickly prohibitive when $r$ is large enough. We also observe a gain in time by using orthogonal functions from order 3.

Experiments are done to compare $P_r$ and the optimal finite element space for pyramids, with the orthogonal base of Proposition 2.2 for $P_r$, which is also the base proposed by Kirby et

Table 4.7: Comparison of iterative, sparse and dense solver used to solve $Mx = y$ for pyramidal elements.

<table>
<thead>
<tr>
<th>Order</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense Solver</td>
<td>91s</td>
<td>144s</td>
<td>183s</td>
<td>295s</td>
<td>340s</td>
<td>503s</td>
<td>652s</td>
</tr>
<tr>
<td>Iterative Solver</td>
<td>150s</td>
<td>150s</td>
<td>149s</td>
<td>331s</td>
<td>366s</td>
<td>403s</td>
<td>439s</td>
</tr>
<tr>
<td>Sparse Solver</td>
<td>74s</td>
<td>101s</td>
<td>121s</td>
<td>148s</td>
<td>178s</td>
<td>227s</td>
<td>235s</td>
</tr>
</tbody>
</table>

Table 4.8: Time spent for 100 iterations of leap-frog scheme for meshes of non-affine pyramids containing one million DOFs for $E_x$.

<table>
<thead>
<tr>
<th>Order</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparse solver</td>
<td>523s</td>
<td>505s</td>
<td>508s</td>
<td>569s</td>
<td>619s</td>
<td>692s</td>
<td>766s</td>
</tr>
<tr>
<td>Warburton trick</td>
<td>460s</td>
<td>432s</td>
<td>411s</td>
<td>451s</td>
<td>471s</td>
<td>494s</td>
<td>548s</td>
</tr>
<tr>
<td>Nodal functions</td>
<td>378s</td>
<td>532s</td>
<td>702s</td>
<td>1135s</td>
<td>2425s</td>
<td>7618s</td>
<td>15350s</td>
</tr>
</tbody>
</table>
The number of degrees of freedom and computational time can be read in Table 4.9.

Table 4.9: Time step, number of DOFs and computational time necessary to reach an error level below one percent for a cubic cavity meshed with affine pyramids. Use of $P_r$ (top) and optimal finite element space (bottom) with orthogonal expansion.

<table>
<thead>
<tr>
<th>Order</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta t$</td>
<td>0.0238</td>
<td>0.0258</td>
<td>0.0248</td>
<td>0.024</td>
<td>0.0216</td>
<td>0.0205</td>
<td>0.0203</td>
</tr>
<tr>
<td>DOFs</td>
<td>960 000</td>
<td>461 370</td>
<td>336 000</td>
<td>258 048</td>
<td>246 960</td>
<td>213 840</td>
<td>165 000</td>
</tr>
<tr>
<td>CPU Time</td>
<td>60 449s</td>
<td>27 564s</td>
<td>22 422s</td>
<td>19 111s</td>
<td>21 433s</td>
<td>22 074s</td>
<td>19 056s</td>
</tr>
</tbody>
</table>

The optimal finite element space is more efficient, even on affine pyramids, since the computational time required for eigth-order is 5 204s for optimal space, and 22 074s for polynomials.

4.4. Tetrahedra

For tetrahedra, the use of orthogonal functions is known to be not very attractive for low orders (Warburton [25]). In Table 4.10, we report computational times obtained for different orders of approximation with both nodal and orthogonal bases. Nodal basis functions provide faster algorithms when $r < 10$. We use quadrature-free algorithms as described in Hesthaven and Warburton [1] to evaluate the integrals.

Table 4.10: Time spent for 100 iterations of leap frog-scheme for meshes containing one million DOFs for $E_x$ with affine tetrahedra

<table>
<thead>
<tr>
<th>Order</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodal</td>
<td>379s</td>
<td>358s</td>
<td>327s</td>
<td>343s</td>
<td>428s</td>
<td>541s</td>
<td>680s</td>
<td>1023s</td>
<td>1751s</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>527s</td>
<td>601s</td>
<td>559s</td>
<td>595s</td>
<td>679s</td>
<td>722s</td>
<td>798s</td>
<td>992s</td>
<td>1074s</td>
</tr>
</tbody>
</table>

5. Applications to Maxwell’s Equations

5.1. Convergence for a Cubic Cavity

To get the convergence order of the different methods tested, we consider the same test-case as in Section 4 with a mesh containing non-affine pyramids (see Fig 1.2). Here, the non-affine pyramids are not tending to affine elements as the mesh size $h$ tends to 0. For example, you could think of trapezoids, whose size is tending to 0, but whose shape is always the same.

Fig. 5.1 displays the solutions for $T = 5$ and $T = 50$. An error analysis is performed for the solution obtained at $T = 50s$ for orders 3 and 5 by using the sparse solver to compute the mass matrix or Warburton’s trick. The reference solution is taken as the numerical solution obtained with hexahedra of order 7. As shows Fig. 5.2, this trick does not provide a $h$-convergent scheme as expected, whereas our method does. However, we may conjecture that Warburton’s
trick does provide a \( p \)-convergent scheme, since the error of consistency decreases when order 5 is used instead of order 3. Moreover, the level of consistency error (less than 2\%) is quite acceptable in most simulations.

![Cube of size 10\( \lambda \) - Solution for \( T = 5 \) (left) and \( T = 50 \) (right)](image)

Fig. 5.1. Cube of size 10\( \lambda \) - Solution for \( T = 5 \) (left) and \( T = 50 \) (right)

![Relative \( L^2 \) error obtained on deformed hybrid mesh (tetras+pyramids) - Cube of size 10\( \lambda \) - Fourth-order low-storage Runge-Kutta scheme](image)

Fig. 5.2. Relative \( L^2 \) error obtained on deformed hybrid mesh (tetras+pyramids) - Cube of size 10\( \lambda \) - Fourth-order low-storage Runge-Kutta scheme

### 5.2. Air Balloon

We consider the scattering of an air balloon in a parallelepiped box \([-250, 50] \times [-130, 180] \times [90, 490]\), with the following source

\[
    f(x, t) = e^{-13.8 \left( \frac{r}{r_0} \right)^2} e^{-0.001(t-t_0)^2} \sin(2\pi f_0 t), \quad r_0 = 15, \; f_0 = 0.08
\]
Table 5.1: Number of degrees of freedom and computational time needed to reach an $L^2$-error of 1% with fifth order approximation, for $T = 432$ - Leap-frog scheme

<table>
<thead>
<tr>
<th>Mesh used</th>
<th>$L^2$-error</th>
<th>Nb of DOFs</th>
<th>Δ$t$</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Nodal</td>
</tr>
<tr>
<td>Hybrid</td>
<td>0.0093</td>
<td>22.4 million</td>
<td>0.032</td>
<td>4d 2h 36min</td>
</tr>
<tr>
<td>Tetrahedral</td>
<td>0.011</td>
<td>37.9 million</td>
<td>0.046</td>
<td>11d 10h 19min</td>
</tr>
</tbody>
</table>

The hybrid mesh used for the experiment, chosen to provide an $L^2$-error below one percent with $Q_5$ approximation, is presented on Fig. 5.3. We compare the results obtained on this hybrid mesh made of 122620 elements (92790 hexahedra (88966 affine), 12223 tetrahedra, 15063 pyramids (4363 affine) and 2544 wedges (1 affine), with the results obtained on a pure tetrahedral mesh made of 677069 affine tetrahedra.

Fig. 5.3. Hybrid mesh used for the computations of the scattering of an air balloon.

For this experiment, we use the classical leap-frog scheme for the time discretization. Because of the presence of small elements, the stability condition (CFL) is quite restrictive, so that we have chosen the time step as the maximal allowable time step. Due to the restrictive stability condition, a second-order leap-frog scheme has been preferred. The error observed here is mainly due to the space approximation, the time error is below 0.1%.

Fig. 5.4 displays the $E_x$ component of the numerical solution obtained for $T = 288$ and $T = 432$. Since the mesh is not curved, we compare each solution to a numerical solution computed on the same mesh but with an order of approximation equal to $r + 1$ instead of $r$. In Table 5.1, we detail the computational time needed to reach an $L^2$ error lower than one percent.

Numerical experiments have been completed on 256 processors. Computational times are obtained by summing the CPU time spent on each processor and subtracting the cost of communications so that the global time is not influenced by problems of load-balancing. The obtained
ideal time would be the same time obtained if the simulation had been launched on a single processor. Besides, the observed parallel efficiency was greater than 80%.

**Conclusion**

Highly efficient pyramidal elements of any order have been constructed for discontinuous Galerkin methods. Numerical experiments conducted with these elements on Maxwell’s equations showed a very good behaviour with hybrid meshes.
References
