Journal of Scientific Computing (2022) 90:78
https://doi.org/10.1007/s10915-021-01750-2

Fast Barycentric-Based Evaluation Over Spectral/*hp* Elements

Edward Laughton[1] · **Vidhi Zala**[2] · **Akil Narayan**[3] · **Robert M. Kirby**[2] · **David Moxey**[4]

Received: 5 March 2021 / Revised: 6 December 2021 / Accepted: 10 December 2021 /
Published online: 10 January 2022

Abstract

As the use of spectral/*hp* element methods, and high-order finite element methods in general, continues to spread, community efforts to create efficient, optimized algorithms associated with fundamental high-order operations have grown. Core tasks such as solution expansion evaluation at quadrature points, stiffness and mass matrix generation, and matrix assembly have received tremendous attention. With the expansion of the types of problems to which high-order methods are applied, and correspondingly the growth in types of numerical tasks accomplished through high-order methods, the number and types of these core operations broaden. This work focuses on solution expansion evaluation at arbitrary points within an element. This operation is core to many postprocessing applications such as evaluation of streamlines and pathlines, as well as to field projection techniques such as mortaring. We expand barycentric interpolation techniques developed on an interval to 2D (triangles and quadrilaterals) and 3D (tetrahedra, prisms, pyramids, and hexahedra) spectral/*hp* element methods. We provide efficient algorithms for their implementations, and demonstrate their effectiveness using the spectral/*hp* element library *Nektar++* by running a series of baseline evaluations against the 'standard' Lagrangian method, where an interpolation matrix is gen-

✉ Edward Laughton
el326@exeter.ac.uk

Vidhi Zala
vidhi.zala@utah.edu

Akil Narayan
akil@sci.utah.edu

Robert M. Kirby
kirby@cs.utah.edu

David Moxey
david.moxey@kcl.ac.uk

[1] Mathematics and Physical Sciences, College of Engineering, University of Exeter, Exeter, UK

[2] Scientific Computing and Imaging Institute and School of Computing, University of Utah, Salt Lake City, UT 84112, USA

[3] Scientific Computing and Imaging Institute and Department of Mathematics, University of Utah, Salt Lake City, UT 84112, USA

[4] Department of Engineering, King's College London, London, UK

erated and matrix-multiplication applied to evaluate a point at a given location. We present results from a rigorous series of benchmarking tests for a variety of element shapes, polynomial orders and dimensions. We show that when the point of interest is to be repeatedly evaluated, the barycentric method performs at worst 50% slower, when compared to a cached matrix evaluation. However, when the point of interest changes repeatedly so that the interpolation matrix must be regenerated in the 'standard' approach, the barycentric method yields far greater performance, with a minimum speedup factor of 7×. Furthermore, when derivatives of the solution evaluation are also required, the barycentric method in general slightly outperforms the cached interpolation matrix method across all elements and orders, with an up to 30% speedup. Finally we investigate a real-world example of scalar transport using a non-conformal discontinuous Galerkin simulation, in which we observe around 6× speedup in computational time for the barycentric method compared to the matrix-based approach. We also explore the complexity of both interpolation methods and show that the barycentric interpolation method requires $\mathcal{O}(k)$ storage compared to a best case space complexity of $\mathcal{O}(k^2)$ for the Lagrangian interpolation matrix method.

Keywords High-order finite elements · Spectral/*hp* elements · Point evaluation · Barycentric interpolation

1 Introduction

The application of high-order numerical methods continues to expand, now spanning aeronautics (e.g., [13]) to biomedical engineering (e.g., [3]), in large part due to two factors: the numerical accuracy combined with low dissipation and dispersion errors they can obtain for certain problem classes [7], and the computational efficiency they can achieve when balancing approximation power against computational density [14]. Most high-order finite element, also called spectral/*hp* element, simulation codes mimic their traditional finite element counterparts in terms of software organization such as the development of local operators that are then "assembled" (either in the strict continuous Galerkin sense or in the weak discontinuous Galerkin sense) into a global system that is advanced in some way [5,6,10]. Tremendous effort has been expended to create optimized elemental operations that evaluate, for instance, the solution expansions at the quadrature points which allows for rapid evaluation of the solution at the points of integration needed for computing the stiffness matrix entries, the forcing terms, and other desirable quantities.

However, in the case of history points (positions in the field at which one wants to track a particular quantity of interest over time) [16], pathlines/streamlines [17], isosurface evaluation, [9] or refinement and mortaring [12], evaluation of solution expansions at arbitrary point locations is required. From the perspective of point evaluation over the entire domain, these operations require two phases: given a point in the domain, first finding the element in which that point resides, and then a fast evaluation of the solution expansion on an individual element. Optimization strategies such as octrees and R-trees have been implemented to accelerate the first of these two tasks (e.g., [8]); however, a concerted effort has not been placed on generalized elemental operations such as arbitrary point evaluation.

The purpose of this work is to address the need for efficient arbitrary point evaluation in high-order (spectral/*hp*) element methods. We are building upon the barycentric polynomial interpolation work of Berrut and Trefethen [1], which has been shown to have a myriad of applications within the polynomial approximation world [11]. In this work, we mathematically generalize barycentric polynomial interpolation to arbitrary simplicial elements, and then demonstrate the effectiveness of this approach on the canonical high-order finite elements shapes in 2D (triangles and quadrilaterals) and 3D (tetrahedra, prisms, pyramids, and hexahedra). The operational efficiency range of our work is designed for polynomial degrees often run within the spectral/*hp* community – from polynomial degree 2 to 10. The algorithms presented herein are also implemented in the publicly available high-order finite element library *Nektar*++ [2,15].

The paper proceeds as follows: In Sects. 2 and 3, we lay out the mathematical details of our work. In Sect. 2, we highlight the one-dimensional building blocks of barycentric interpolation and provide the mathematical framework for considering interpolation over finite element shapes generated through successive application of Duffy transformations of an orthotope element, and then in Sect. 3 we provide details on expanding these ideas to both tensor-product constructed 2D and 3D elements as used in *Nektar*++. In Sect. 5, we present both algorithmic and implementation details. We provide details that facilitate reproducibility of our results as well as complexity and storage analysis of the proposed strategy. In Sect. 6, we present various test cases that demonstrate the efficacy and efficiency of our proposed work, as well as a capstone example that highlights a real-world application of our strategy. We summarize our contributions and conclude with possible future work in Sect. 7.

2 Univariate Barycentric Interpolation

Berrut and Trefethen [1] propose barycentric Lagrange interpolation for stable, efficient evaluation of polynomial interpolants. With $\eta \in [-1, 1]$ the independent variable, let $\{z_j\}_{j=0}^{k} \subset [-1, 1]$ denote $k + 1$ unique nodes. Defining Q_k as the space of polynomials of degree k or less in one variable, then any $p \in Q_k$ can be represented in its barycentric form,

$$p(\eta) = \frac{\sum_{j=0}^{k} \frac{w_j p_j}{(\eta - z_j)}}{\sum_{j=0}^{k} \frac{w_j}{(\eta - z_j)}} =: \frac{N(\eta)}{D(\eta)}, \qquad p_j := p(z_j), \tag{1}$$

where the weights w_j are given by

$$w_j = \frac{1}{\prod_{i=0, i\neq j}^{k} (z_i - z_j)} \qquad \forall j = 0, 1, \cdots, k. \tag{2}$$

The weights are independent of the polynomial p, and depend only on the nodal configuration. Barycentric form (1) reveals that given the values $\{p_j\}_{j=0}^{k}$ of a polynomial, then evaluation of p at an arbitrary location η can be accomplished without solving linear systems or evaluating cardinal Lagrange interpolants.

In this paper, we focus on the algorithmic advantages of using barycentric form, with proper extensions to some standard multivariate non-tensorial domains that are popular in high-order (spectral/*hp*) finite element methods. These algorithmic advantages stem from the univariate algorithmic advantages: The map $\eta \mapsto p(\eta)$ using the formula (1) requires $\mathcal{O}(k)$ arithmetic operations, whereas the same map using standard linear expansions frequently requires $\mathcal{O}(k^2)$ operations.

Throughout our discussion, we consider the nodes z_j, and subsequently, through (2), the barycentric weights w_j, as given and fixed. To emphasize the $\mathcal{O}(k)$ complexity of the operations, which we consider in the following section, we define

$$S_r(\boldsymbol{v}, \eta) := \sum_{j=0}^{k} \frac{v_j w_j}{(\eta - z_j)^r}, \qquad \boldsymbol{v} = (v_0, \ldots, v_k)^T, \tag{3}$$

which, given r, $\boldsymbol{v}$, ostensibly requires only $\mathcal{O}(k)$ complexity to evaluate $\eta \mapsto S_r(\boldsymbol{v}, \eta)$. Note in particular that $S_r(\boldsymbol{v}, \eta)$ is linear in $\boldsymbol{v}$ and that

$$\frac{\mathrm{d}}{\mathrm{d}\eta} S_r(\boldsymbol{v}, \eta) = -r S_{r+1}(\boldsymbol{v}, \eta). \tag{4}$$

We can write (1) as

$$p(\eta) = \frac{S_1(\boldsymbol{p}, \eta)}{S_1(\mathbf{1}, \eta)}, \qquad \boldsymbol{p} = (p_0, \ldots, p_k)^T, \qquad \mathbf{1} := (1, \ldots, 1)^T \in \mathbb{R}^n, \tag{5}$$

which clearly demonstrates the $\mathcal{O}(k)$ complexity if $\boldsymbol{p}$ is furnished.

2.1 Alternatives to Barycentric Form

Consider a linear expansion representation of $p \in Q_k$, written in terms of either an (arbitrary) basis $\{\phi_j\}_{j=0}^k$ of Q_k, or the cardinal Lagrange functions of the interpolation points $\{z_j\}_{j=0}^k$:

$$p(\eta) = \sum_{j=0}^{k} c_j \phi_j(\eta), \qquad \operatorname{span}\{\phi_0, \ldots \phi_k\} = Q_k, \tag{6}$$

where the ϕ_j basis functions can be, e.g., cardinal Lagrange interpolants, monomials, or orthogonal polynomials:

$$\phi_j(\eta) = \prod_{\substack{i=0,\ldots,k \\ i \neq j}} \frac{\eta - z_i}{z_j - z_i}, \tag{7a}$$

$$\phi_j(\eta) = \eta^{j-1}, \tag{7b}$$

$$\phi_j(\eta) = \psi_j(\eta), \tag{7c}$$

with ψ_j any orthogonal polynomial family, such as Legendre or Chebyshev polynomials. Our main focus is the computational complexity of evaluating a polynomial interpolant given the data values $\{p_j\}_{j=0}^k$. The following diagram summarizes the complexity of utilizing each of these procedures to accomplish the evaluation $\eta \mapsto p(\eta)$:

$$\begin{tikzcd}
\text{Barycentric (1 :} & \{(z_j)\}_{j=0}^k, \eta \arrow[r, "\mathcal{O}(k^2)"] & \{(w_j, z_j, p_j)\}_{j=0}^k, \eta \arrow[r, "\mathcal{O}(k)"] & p(\eta) \\
\text{Monomial (6), (7b) :} & \{(z_j, p_j)\}_{j=0}^k, \eta \arrow[r, "\mathcal{O}(k^3)"] & \{(c_j)\}_{j=0}^k, \eta \arrow[r, "\substack{\text{Naive: } \mathcal{O}(k^2) \\ \text{Horner: } \mathcal{O}(k)}"] & p(\eta) \\
\text{Orth. Poly. (6), (7c) :} & \{(z_j, p_j)\}_{j=0}^k, \eta \arrow[r, "\mathcal{O}(k^3)"] & \{(c_j)\}_{j=0}^k, \eta \arrow[r, "\substack{\text{Naive: } \mathcal{O}(k^2) \\ \text{Clenshaw: } \mathcal{O}(k)}"] & p(\eta) \\
\text{Lagrange (6), (7a) :} & & \{(z_j, p_j)\}_{j=0}^k, \eta \arrow[r, "\mathcal{O}(k^2)"] & p(\eta)
\end{tikzcd} \tag{8}$$

Above, we have alluded to the fact that direct evaluation of monomial or orthogonal polynomial $(k+1)$-term expansions appears to require $\mathcal{O}(k^2)$ complexity, but clever rearrangement of elements in the summation can lower this to $\mathcal{O}(k)$ in both cases, using either Horner's algorithm (monomials) or Clenshaw's algorithm [4] (orthogonal polynomials). Viewed in this way, there are two advantages to utilizing the barycentric form. The initial one-time computation for the barycentric weights is cheaper than for the monomials. Furthermore, with fixed nodal locations z_j, the weights w_j need not be recomputed if p is changed; in other words, the weights w_j do not depend on p. Hence if k and the nodes z_j are given and fixed, then the weights w_j can be precomputed and (re-)used for *any* $p \in Q_k$.

Our first task in this paper is to generalize the barycentric procedure to first- and second-derivative.

2.2 Barycentric Evaluation of Derivatives

A formula for the derivative of a polynomial can be derived from the barycentric form (1) that inherits its advantages. We first note two auxiliary expressions for the derivative of the numerator and denominator rational functions,

$$N'(\eta) = -S_2(\boldsymbol{p}, \eta), \qquad D'(\eta) = -S_2(\mathbf{1}, \eta), \tag{9}$$

each of which ostensibly also requires only $\mathcal{O}(k)$ complexity to evaluate if the weights w_j are precomputed. Then, by directly differentiating (1), we have

$$p'(\eta) = \frac{N'(\eta) - p(\eta)D'(\eta)}{D(\eta)} = \frac{S_2\left(p(\eta) - \boldsymbol{p}, \eta\right)}{S_1(\mathbf{1}, \eta)}, \tag{10}$$

where the vector $p(\eta) - \boldsymbol{p}$ has entries $p(\eta) - p_j$. Therefore, this “barycentric" form for $p'(\eta)$ requires (i) an evaluation of $p(\eta)$ that can be accomplished in $\mathcal{O}(k)$ complexity using (1), and (ii) an additional $\mathcal{O}(k)$ evaluation of the summation above. Thus, $\eta \mapsto p'(\eta)$ can be evaluated with only $\mathcal{O}(k)$ complexity.

A similar computation shows that

$$p''(\eta) = \frac{2}{S_1(\mathbf{1}, \eta)} \left(p'(\eta) S_2(\mathbf{1}, \eta) - S_3(p(\eta) - \boldsymbol{p}, \eta)\right). \tag{11}$$

Again, since $\eta \mapsto p(\eta)$ and $\eta \mapsto p'(\eta)$ can be evaluated with $\mathcal{O}(k)$ effort through (1) and (10), some extra $\mathcal{O}(k)$ effort to evaluate S_2 and S_3 above yields an evaluation $\eta \mapsto p''(\eta)$ that can also be accomplished in $\mathcal{O}(k)$ time.

3 Tensorization of the Barycentric Form

All the evaluations considered above can be generalized to tensorial formulations, which is the main topic of this section. To that end, we introduce some multidimensional notations. Let $\boldsymbol{\eta} := (\eta_1, \ldots, \eta_d) \in [1, 1]^d$ be a d-dimensional vector. Given some multi-index $\boldsymbol{k} \in \mathbb{N}_0^d$, we introduce the tensorial space of polynomials $Q_{\boldsymbol{k}}$ defined by $\boldsymbol{k}$:

$$Q_{\boldsymbol{k}} := \operatorname{span}\left\{\boldsymbol{\eta}^{\boldsymbol{j}} \;\middle|\; \boldsymbol{j} \in \mathbb{N}_0^d \text{ and } \boldsymbol{j} \leq \boldsymbol{k}\right\}, \tag{12}$$

where we have adopted the standard multi-index notation,

$$\boldsymbol{\eta}^{\boldsymbol{j}} := \prod_{q=1}^{d} \eta_q^{j_q}, \qquad \boldsymbol{j} = (j_1, \ldots, j_d) \in \mathbb{N}_0^d, \tag{13}$$

and $\boldsymbol{j} \le \boldsymbol{k}$ is true if all the component-wise inequalities are true.

Fixing $\boldsymbol{k}$, we consider the case of representing an element p of $Q_{\boldsymbol{k}}$ through its values on a discrete tensorial grid of size $\prod_{q=1}^{d} k_q$. Like the univariate case, linear expansions in cardinal Lagrange, monomial, and/or orthogonal polynomials are common, but we will exercise the barycentric form. Let a tensorial grid on a $[-1, 1]^d$ orthotope be given, with $k_q + 1$ points in direction q:

$$\left\{z_{j,q}\right\}_{j=0}^{k_q} \subset [-1, 1], \tag{14}$$

for $q = 1, \ldots, d$. The tensorization of these grids results in the multidimensional grid $\left\{\boldsymbol{\eta}_{\boldsymbol{j}}\right\}_{\boldsymbol{j} \le \boldsymbol{k}}$ defined by

$$\boldsymbol{z}_{\boldsymbol{j}} := \left(z_{j_1,1},\, z_{j_2,2},\, \ldots,\, z_{j_d,d}\right) \in [-1, 1]^d. \tag{15}$$

Given this tensorial configuration of nodes, we define univariate barycentric weights associated with each dimension in a fashion similar to (2),

$$w_{q,\ell} := \frac{1}{\prod\limits_{i=0, i \ne j}^{k} (z_{i,q} - z_{j,q})}, \qquad 0 \le j \le k_q, \qquad 1 \le q \le d. \tag{16}$$

Then, given the data

$$\left\{p_{\boldsymbol{j}}\right\}_{\boldsymbol{j} \le \boldsymbol{k}}, \qquad p_{\boldsymbol{j}} := p\left(\boldsymbol{z}_{\boldsymbol{j}}\right), \tag{17}$$

for $p \in Q_{\boldsymbol{k}}$, then the multidimensional barycentric form of p is

$$p(\boldsymbol{\eta}) = \frac{\sum_{\boldsymbol{j} \le \boldsymbol{k}} \frac{p_{\boldsymbol{j}} w_{\boldsymbol{j}}}{\odot(\boldsymbol{\eta} - \boldsymbol{z}_{\boldsymbol{j}})}}{\sum_{\boldsymbol{j} \le \boldsymbol{k}} \frac{w_{\boldsymbol{j}}}{\odot(\boldsymbol{\eta} - \boldsymbol{z}_{\boldsymbol{j}})}}, \quad \odot\left(\boldsymbol{\eta} - \boldsymbol{z}_{\boldsymbol{j}}\right) := \prod_{q=1}^{d} \left(\eta_q - z_{j_q,q}\right), \quad w_{\boldsymbol{j}} := \prod_{q=1}^{d} w_{q,j_q}. \tag{18}$$

Given $\boldsymbol{y} \in [-1, 1]$, an evaluation of $p(\boldsymbol{y})$ above requires $\mathcal{O}\left(\prod_{q=1}^{d} k_q\right)$ operations, corresponding to the complexity of the summations.

3.1 Dimension-by-Dimension Approach

Instead of the direct approach (18) for evaluating p, we utilize a dimension-by-dimension computation that is slightly more computationally expensive but allows us to directly leverage univariate procedures, greatly simplifying the software implementation. First, consider the functions $\widetilde{p}_q$, for $q = d - 1, \ldots, 1$, each of which is a function of $\eta_1, \ldots, \eta_q$ formed by freezing $\eta_s = y_s$ for $s > q$:

$$\widetilde{p}_d := p, \quad \widetilde{p}_q(\eta_1, \ldots, \eta_q) := p\left(\eta_1, \ldots, \eta_q, y_{q+1}, \ldots, y_d\right) = \widetilde{p}_{q+1}(\eta_1, \ldots, \eta_q, y_{q+1}). \tag{19}$$

In order to evaluate $p(\boldsymbol{y})$, we will proceed by iteratively constructing $\widetilde{p}_q$ from $\widetilde{p}_{q+1}$ for $q = d-1, \ldots, 1$. Via barycentric form, “constructing" $\widetilde{p}_q$ amounts to evaluating this function on the q-dimensional tensorial grid

$$\widetilde{z}_{q,\boldsymbol{j}} = \left(z_{j_1,1},\, z_{j_2,2},\, \ldots,\, z_{j_q,q}\right) \in [-1,1]^q. \tag{20}$$

Then, for example, to first generate $\widetilde{p}_{d-1}$, we must evaluate

$$\widetilde{p}_{d-1}\left(\widetilde{z}_{d-1,\boldsymbol{j}}\right) = p\left(z_{j_1,1}, \ldots, z_{j_{d-1},d-1}, y_d\right), \tag{21}$$

for every $\boldsymbol{j} \leq (k_1, \ldots, k_{d-1})$. This can be accomplished via univariate procedures in $\mathcal{O}(k_d)$ complexity since, for each fixed $\boldsymbol{j}$,

$$y_d \mapsto p\left(z_{j_1,1}, \ldots, z_{j_{d-1},d-1}, y_d\right), \tag{22}$$

is a polynomial of degree k_d, and hence obeys the barycentric formula,

$$p\left(z_{j_1,1}, \ldots, z_{j_{d-1},d-1}, y_d\right) = \frac{S_1\left(\widetilde{\boldsymbol{p}}_{\boldsymbol{j}}, y_d\right)}{S_1\left(\mathbf{1}, y_d\right)}, \qquad \widetilde{\boldsymbol{p}}_{\boldsymbol{j}} := \left(p_{(j_1,\ldots,j_{d-1},\ell)}\right)_{\ell=1}^{k_d}. \tag{23}$$

In this way, we proceed to iteratively construct $\widetilde{p}_q$, amounting to $\prod_{j=1}^{q} k_j$ evaluations, each of computational complexity $\mathcal{O}(k_{q+1})$,

$$\left.\begin{array}{lcll} p & \xrightarrow{\prod_{j=1}^{d-1} k_j \mathcal{O}(k_d) \text{ evaluations}} & \widetilde{p}_{d-1} & \\ \widetilde{p}_q & \xrightarrow{\prod_{j=1}^{q-1} k_j \mathcal{O}(k_q) \text{ evaluations}} & \widetilde{p}_{q-1} & (2 < q < d) \\ \widetilde{p}_1 & \xrightarrow{1\mathcal{O}(k_1) \text{ evaluation}} & p(\boldsymbol{y}) & \end{array}\right\} \tag{24}$$

In summary, computing $\boldsymbol{y} \mapsto p(\boldsymbol{y})$ can be accomplished with the procedure above, which entails repeated use of the univariate barycentric form (5).

As mentioned earlier, the cost of the procedure (24) is slightly more expensive than direct evaluation of the multidimensional barycentric form (18). In particular, the ($\boldsymbol{k}$-asymptotic) cost of the direct evaluation (18) is $\prod_{q=1}^{d} k_q$. On the other hand, the dimension-by-dimension approach (24) incurs additional lower-order costs, and has complexity scaling as,

$$\prod_{q=1}^{d} k_q + \prod_{q=1}^{d-1} k_q + \ldots = \sum_{j=1}^{d} \prod_{q=1}^{j} k_q \leq d \prod_{q=1}^{d} k_q, \tag{25}$$

where the inequality is a very crude bound. Thus, while the algorithm described by (24) is formally more expensive than direct evaluation (18), the actual additional cost is relatively small. In particular, for the physically relevant cases of $d = 2, 3$, this minor increase in cost is acceptable for the achieved gain in implementation ease.

3.2 Tensorial Functions

We end this section with a brief remark on a direct simplification that can be employed in the special case that one has prior knowledge that the polynomial p is tensorial, i.e., of the form,

$$p(\boldsymbol{\eta}) = \prod_{j=1}^{d} p_j(\eta_j), \tag{26}$$

for some univariate polynomials $\{p_j\}_{j=1}^{d}$ satisfying $\deg p_j \leq k_j$. In many high-order FEM simulations, basis functions in $\boldsymbol{\eta}$ space are often tensorial polynomials, so that this situation does occur in practice. In this tensorial case, computing the barycentric weights is simpler since we need to only compute the k_j *univariate* weights for dimension j associated with the grid $z_{q,j}$ for $q \in [k_j]$. Thus, we need to only compute $\sum_{j=1}^{d} k_j$ weights, as opposed to the full set of $\prod_{j=1}^{d} k_j$ multivariate weights associated with (18).

Evaluating $\boldsymbol{\eta} \mapsto p(\boldsymbol{\eta})$ is likewise faster in this case: once the univariate weights are computed, then each univariate barycentric evaluation $\eta_j \mapsto p_j(\eta_j)$ requires $\mathcal{O}(k_j)$ complexity. Therefore, $\boldsymbol{\eta} \mapsto p(\boldsymbol{\eta})$ requires only $\mathcal{O}(\sum_{j=1}^{d} k_j)$ complexity, as opposed to the full multivariate $\mathcal{O}(\prod_{j=1}^{d} k_j)$ complexity.

3.3 Derivatives

Section 3.1 discusses how we accomplish the evaluation $\boldsymbol{\eta} \mapsto p(\boldsymbol{\eta})$ algorithmically by iteratively evaluating along each dimension. This procedure is directly extensible to evaluation of (Cartesian) partial derivatives. For example, if $p \in Q_{\boldsymbol{k}}$, suppose for some multi-index $\boldsymbol{\lambda} \in \mathbb{N}_0^d$ we wish to evaluate the order-$\boldsymbol{\lambda}$ derivative,

$$p^{(\boldsymbol{\lambda})} = \frac{\partial^{|\boldsymbol{\lambda}|} p}{\partial \boldsymbol{\eta}^{\boldsymbol{\lambda}}}, \qquad \partial \boldsymbol{\eta}^{\boldsymbol{\lambda}} = \partial x_1^{\lambda_1} \partial x_2^{\lambda_2} \cdots \partial x_d^{\lambda_d}. \tag{27}$$

We are mostly concerned with $|\boldsymbol{\lambda}| \leq 2$, i.e., at most “second" derivatives, but the procedure we describe applies to derivatives of arbitrary order. The dimension-by-dimension approach can be accomplished with essentially the same procedure as articulated in Section 3.1: Define

$$\begin{aligned} \widetilde{p}_d &:= p, \\ \widetilde{p}_q(\eta_1, \ldots, \eta_q) &:= \frac{\partial \widetilde{p}_{q+1}}{\partial \eta_{q+1}^{\lambda_{q+1}}}(\eta_1, \ldots, \eta_q, y_{q+1}), \end{aligned} \tag{28}$$

so that constructing $\widetilde{p}_q$ from $\widetilde{p}_{q+1}$ at a single grid point requires evaluation of the order-λ_{q+1} derivative along dimension $q+1$. Through procedures outlined in Sect. 2.2, we can accomplish this in $\mathcal{O}(k_{q+1})$ complexity. We must evaluate the derivative at each of the $\prod_{j=1}^{q} k_j$ grid points associated with dimensions $1, \ldots, q$. Therefore, the outline and complexity of this procedure is precisely as given in (24), except that $p(\boldsymbol{y})$ should be replaced by $p^{(\boldsymbol{\lambda})}(\boldsymbol{y})$.

4 Nontensorial Multidimensional Formulations Via Duffy Transformations

The goal of this section is to describe how barycentric interpolation in the tensorial case over d-dimensional orthotopes of Sect. 3 can be utilized for efficient evaluation of polynomial approximations in certain nontensorial cases. We focus on (potentially) non-tensorial polynomial approximations in a d-dimensional variable $\boldsymbol{\xi}$. The variable $\boldsymbol{\xi}$ will be related to the tensorial variable $\boldsymbol{\eta}$ through “collapsed coordinates" effected by a Duffy transformation, as described in Sect. 4.1. The transformation can be applied to a variety of common FEM element types, see Table 1. A description of how barycentric evaluation procedures can be used to evaluate polynomials on these potentially nontensorial geometries is given in Sect. 4.2; specifically, the procedure is given by (32). That section also gives precise conditions to which polynomials space p must belong so that the evaluation is exact (Theorem 4.1). Section 4.3 specializes the evaluation exactness conditions to common element types used

Table 1 Multidimensional domains resulting from collapsed coordinates, multi-indices $\boldsymbol{a}$ and $\boldsymbol{b}$ identifying the associated multivariate Duffy map, and ancestor functions $g_{\boldsymbol{a},\boldsymbol{b}}$ defined in (33)

	Geometric region $E_{\boldsymbol{a},\boldsymbol{b}}$	c	$\left\{(a_j, b_j)\right\}_{j=1}^{c}$	$g = g_{\boldsymbol{a},\boldsymbol{b}}$ evaluations
$d = 2$	Quadrilateral	0	—	$g(1) = \{1\}, g(2) = \{2\}$
	Triangle	1	$(1, 2)$	$g(1) = \{1\}, g(2) = \{1, 2\}$
$d = 3$	Hexahedron	0	—	$g(1) = \{1\}, g(2) = \{2\}, g(3) = \{3\}$
	Prism	1	$(1, 2)$	$g(1) = \{1\}, g(2) = \{1, 2\}, g(3) = \{3\}$
	Tetrahedron	2	$(1, 2), (2, 3)$	$g(1) = \{1\}, g(2) = \{1, 2\}, g(3) = \{1, 2, 3\}$
	Pyramid	2	$(1, 3), (2, 3)$	$g(1) = \{1\}, g(2) = \{2\}, g(3) = \{1, 2, 3\}$

in the spectral/hp community. Section 4.4 closes the section by discussing extension of the evaluation routines to derivative evaluations through use of the chain rule.

4.1 Collapsed Coordinates

We consider polynomials in d variables $\boldsymbol{\xi}$. The particular type of $\boldsymbol{\xi}$-polynomials we consider are defined via a mapping from $\boldsymbol{\eta}$ space to $\boldsymbol{\xi}$ space, where $\boldsymbol{\eta} \in [-1, 1]^d$ is the tensorial variable considered in Sect. 3. The essential building block that allows us to specify the $\boldsymbol{\eta} \leftrightarrow \boldsymbol{\xi}$ relationship is the Duffy transformation, which is a variable transformation in two dimensions. For $\boldsymbol{\eta} \in [-1, 1]^d$ and some fixed $i, j \in \{1, \ldots, d\}$ with $i \neq j$, define $D_{i,j}$ as the Duffy transformation that "collapses" dimension i with respect to or along dimension j and is the identity map on all the other dimensions,

$$\boldsymbol{\zeta} = (\zeta_1, \ldots, \zeta_d) := D_{i,j}(\boldsymbol{\eta}), \qquad \zeta_\ell = \begin{cases} \frac{1}{2}(1 + \eta_\ell)\left(1 - \eta_j\right) - 1, & \ell = i, \\ \eta_\ell, & \ell \neq i \end{cases}, \tag{29}$$

for $\ell = 1, \ldots, d$.

Various domains in d dimensions can be created by composing Duffy maps. For composing $c < d$ maps, we let $\boldsymbol{a} \in [d]^c$ have components that represent the dimensions that are collapsed by a Duffy transformation, and let $\boldsymbol{b} \in [d]^c$ have components specifying along which dimensions the collapse occurs. We place the following restrictions on the entries of $\boldsymbol{a}$ and $\boldsymbol{b}$:

- $a_\ell < b_\ell$ for $\ell = 1, \ldots, c$
- $a_\ell < b_\ell$ for $\ell = 1, \ldots, c$
- $a_\ell < a_{\ell+1}$ for $\ell = 1, \ldots, c - 1$.

We now define the variable $\boldsymbol{\xi}$ as the image of $\boldsymbol{\eta}$ under a composition of Duffy transformations defined by $\boldsymbol{a}$ and $\boldsymbol{b}$,

$$\boldsymbol{\xi} := D_{\boldsymbol{a},\boldsymbol{b}} := D_{a_1,b_1} \circ D_{a_2,b_2} \circ \cdots \circ D_{a_c,b_c}(\boldsymbol{\eta}), \qquad E(\boldsymbol{a}, \boldsymbol{b}) := D_{\boldsymbol{a},\boldsymbol{b}}\left([-1, 1]^d\right). \tag{30}$$

We are interested primarily in these domains for dimensions $d = 2, 3$. Table 1 illustrates various standard geometric domains that are the result of particular choices of coordinate collapses.

A visual example with $d = 2$ is also shown in Fig. 1 which gives the Duffy transformations between reference triangles and reference quadrilaterals.

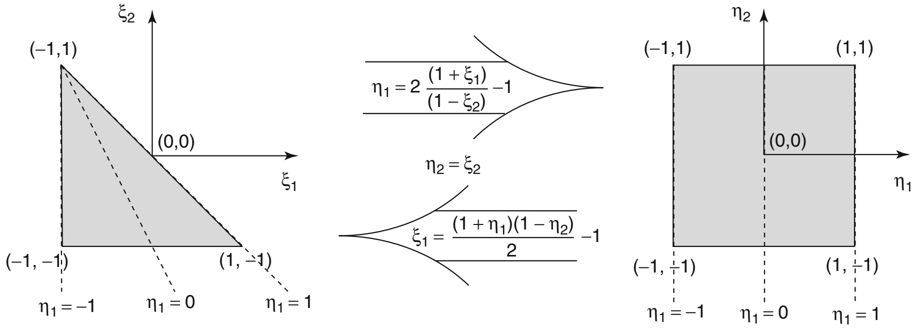

Fig. 1 Duffy transformations between triangles and quadrilateral reference elements

4.2 Evaluation of Polynomials

The previous section illustrates how various standard domains that are used to tesselate space in finite element simulations are constructed. This section considers how we can employ barycentric evaluation in $\boldsymbol{\eta}$ space to accomplish evaluation of polynomials in $\boldsymbol{\xi}$ space. In what follows we assume that the dimension d, the grid size multi-index $\boldsymbol{k}$, and the Duffy transformation parameters c, $\boldsymbol{a}$, and $\boldsymbol{b}$ are all given and fixed.

Recall that on the tensorial domain $\boldsymbol{\eta} \in [-1, 1]^d$ we have a tensorial grid $Z_{\boldsymbol{k}}$ comprised of k_q points in dimension q, resulting in a total of $\prod_{q=1}^{d} k_q$ points in $Z_{\boldsymbol{k}}$. The image of this grid in $\boldsymbol{\xi}$ space is the result of applying the Duffy transformation:

$$\boldsymbol{y}_j := D_{\boldsymbol{a},\boldsymbol{b}}\left(z_j\right), \qquad\qquad \boldsymbol{j} \le \boldsymbol{k}. \tag{31a}$$

Assume that data values are furnished from a given function p,

$$P_{\boldsymbol{k}} := \left(p_{\boldsymbol{j}}\right)_{\boldsymbol{j}\le\boldsymbol{k}}, \qquad\qquad p_{\boldsymbol{j}} := p\left(\boldsymbol{y}_{\boldsymbol{j}}\right), \tag{31b}$$

and are provided on the grid $\boldsymbol{y}_{\boldsymbol{j}}$. Naturally, these values can be considered as data values in $\boldsymbol{\eta}$ space under the (inverse) Duffy transformation,

$$\left\{\left(z_{\boldsymbol{j}}, p_{\boldsymbol{j}}\right)\right\}_{\boldsymbol{j}\le\boldsymbol{k}},$$

and therefore the barycentric routines developed in tensorial form in Sect. 3 can be applied. The main result of this section is the provision of conditions on p in $\boldsymbol{\xi}$ space under which applying the tensorial barycentric interpolation procedure in $\boldsymbol{\eta}$ space results in an exact evaluation. More precisely, we consider the following algorithmic set of steps given a point $\boldsymbol{\xi}$ in $E(\boldsymbol{a}, \boldsymbol{b})$, and a function p:

$$(\boldsymbol{\xi}, p) \xrightarrow{(31)} (\boldsymbol{\xi}, Z_{\boldsymbol{k}}, P_{\boldsymbol{k}}) \xrightarrow{\boldsymbol{\eta}=D_{\boldsymbol{a},\boldsymbol{b}}^{-1}(\boldsymbol{\xi})} (\boldsymbol{\eta}, Z_{\boldsymbol{k}}, P_{\boldsymbol{k}}) \xrightarrow{\text{Section 3}} p(\boldsymbol{\xi}). \tag{32}$$

Thus, our main result below gives conditions on p so that the output on the right of (32) equals the correct evaluation $p(\boldsymbol{\xi})$. To proceed, we need a more involved deconstruction of the element identifier multi-indices $\boldsymbol{a}$ and $\boldsymbol{b}$. The particular rules in Sect. 4.1 that define the possible values of $\boldsymbol{a}$ and $\boldsymbol{b}$ ensure that a collection of tree structures can be constructed from $\boldsymbol{a}$ and $\boldsymbol{b}$. Let each dimension $1, 2, \ldots, d$, correspond to a node. The directed edges correspond to drawing an arrow from node b_j ending at node a_j for each $j = 1, \ldots, c$. Since the indices $\{a_j\}_{j=1}^{c}$ are all distinct, one arrow at most lands at each node, and therefore this construction forms a collection of trees. With this structure, we now define an 'ancestor function' on the set of dimensions, which identifies which indices are ancestors of any dimension,

$$g_{\boldsymbol{a},\boldsymbol{b}} : [d] \to 2^{[d]}, \qquad g_{\boldsymbol{a},\boldsymbol{b}}(q) := \{q\} \bigcup \left\{ i \in [d] \;\middle|\; i \text{ is an ancestor of } q \right\}, \tag{33}$$

where $2^{[d]}$ denotes the power set (set of subsets) of $[d]$. Note that we have also included the index q in $g_{\boldsymbol{a},\boldsymbol{b}}(q)$, so that $g_{\boldsymbol{a},\boldsymbol{b}}(q)$ is always non-empty. The identification of $\boldsymbol{a}$ and $\boldsymbol{b}$ for typical geometries in $d = 2, 3$ is given in Table 1. Finally, through the identification of the relation g_T, we can articulate which functions in $\boldsymbol{\xi}$ space are exactly evaluated via the barycentric form in $\boldsymbol{\eta}$ space.

Theorem 4.1 *With d, $\boldsymbol{k}$, c, $\boldsymbol{a}$, and $\boldsymbol{b}$ all given and fixed, define the following multi-index set:*

$$A := \left\{ \boldsymbol{\alpha} \in \mathbb{N}_0^d \;\middle|\; \sum_{j \in g_{\boldsymbol{a},\boldsymbol{b}}(q)} \alpha_j \le k_q \textit{ for every } q \in [d] \right\}, \tag{34}$$

which defines a polynomial space,

$$P = P(A) := \left\{ \boldsymbol{\xi}^{\boldsymbol{\alpha}} \;\middle|\; \boldsymbol{\alpha} \in A \right\} \subset Q_{\boldsymbol{k}}. \tag{35}$$

Then, for every $p \in P$ (a polynomial in $\boldsymbol{\xi}$ space), the procedure in (32) that utilizes the barycentric evaluation algorithm of Sect. 3 exactly evaluates $p(\xi)$.

Proof Let $p \in P$, and let $\boldsymbol{\alpha} \in \mathbb{N}_0^d$ denote the degree of p, i.e., the polynomial degree of p in dimension q is α_q. Since $p(\boldsymbol{\xi}) = p(\boldsymbol{D}_{\boldsymbol{a},\boldsymbol{b}}(\boldsymbol{\eta}))$, then the result is proven if we can show that $\hat{p} := p \circ \boldsymbol{D}_{\boldsymbol{a},\boldsymbol{b}} \in Q_{\boldsymbol{k}}$, since the barycentric form in $\boldsymbol{\eta}$ space is exact on this space of polynomials. Note that each Duffy transformatiom $\boldsymbol{D}_{\boldsymbol{a},\boldsymbol{b}}$ defined through (29) and (30) is a (multivariate) polynomial, so that $\hat{p} = p \circ \boldsymbol{D}_{\boldsymbol{a},\boldsymbol{b}}$ is a polynomial, and we need to show that only its maximum degree in dimension q is less than or equal to k_q for every $q = 1, \ldots, d$.

Fixing $q \in [d]$, the degree of $\hat{p}$ in dimension q is discernible from the ancestor function $g_{\boldsymbol{a},\boldsymbol{b}}$. Let $\deg_q(f)$ denote the dimension-q degree of a polynomial f. Then the Duffy transformation definition (29) implies that for any two distinct dimensions i, j,

$$\deg_j\left(D_{i,j} \circ f\right) = \deg_i(f) + \deg_j(f), \quad \deg_q\left(D_{i,j} \circ f\right) = \deg_q(f), \quad q \in [d] \backslash \{i\}. \tag{36}$$

Since $\boldsymbol{D}_{\boldsymbol{a},\boldsymbol{b}}$ is a composition of univariate Duffy maps, the degree of $\hat{p}$ along any dimension q can be determined by tracing the history of which dimensions collapse onto q, i.e., is determined by $g_{\boldsymbol{a},\boldsymbol{b}}(q)$. Thus,

$$\deg_q\left(\hat{p}\right) = \deg_q(p) + \sum_{i \text{ is an ancestor of } q} \deg_i(p) = \sum_{i \in g_{\boldsymbol{a},\boldsymbol{b}}(q)} \deg_i(p) = \sum_{i \in g_{\boldsymbol{a},\boldsymbol{b}}(q)} \alpha_i. \tag{37}$$

By assumption on the index set A to which $\boldsymbol{\alpha}$ belongs, this last term is bounded by k_q. □

Given a tensorial grid in $Z_{\boldsymbol{k}}$ in $\boldsymbol{\eta}$ space, Theorem 4.1 precisely describes what type of $\boldsymbol{\xi}$-polynomial space membership p should have so that the procedure (32) exactly evaluates p.

4.3 Specializations

This section describes certain specializations of the apparatus in the previous section. Our specializations will be the two- and three-dimensional domains shown in Table 1. The goal is to show how the exactness condition of Theorem 4.1 manifests on these domains, in particular, to articulate the polynonmial space P defined in (35) on which the barycentric evaluation procedure (32) is exact. We will describe P for a given degree index $\boldsymbol{k}$, and will also present a special "isotropic" case when the number of points is the same in every dimension, i.e., when $\boldsymbol{k} = (k, k, \ldots, k)$ for some non-negative scalar integer k.

4.3.1 Quadrilaterals

We consider $d = 2$, with $c = 0$ Duffy maps. In this case, we have $\boldsymbol{\eta} = \boldsymbol{\xi}$, and both variables take values on $[-1, 1]^2$. Then, given degree $\boldsymbol{k} = (k_1, k_2)$, the set A in (4.1) corresponds to all multi-indices $\boldsymbol{j}$ satisfying $\boldsymbol{j} \leq \boldsymbol{k}$. Therefore, the polynomial space P in (35) is equal to $Q_{\boldsymbol{k}}$,

$$\begin{aligned} P &= \operatorname{span}\left\{\xi_1^{j_1}\xi_2^{j_2} \mid \boldsymbol{j} \leq \boldsymbol{k}\right\} = Q_{\boldsymbol{k}}, \\ P &= \operatorname{span}\left\{\xi_1^{j_1}\xi_2^{j_2} \mid \boldsymbol{j} \leq \boldsymbol{k}\right\} = Q_{(k,k)}, \quad (k = k_1 = k_2). \end{aligned} \tag{38}$$

4.3.2 Triangles

As with quadrilateral elements we have $d = 2$, but we now take $c = 1$, and a Duffy transformation collapse defined by $\boldsymbol{a} = 1$, $\boldsymbol{b} = 2$. Then, the $\boldsymbol{\eta} \leftrightarrow \boldsymbol{\xi}$ map is given by

$$\xi_1 = \frac{1}{2}(1 + \eta_1)(1 - \eta_2), \qquad \xi_2 = \eta_2. \tag{39}$$

The constraints in the definition of A are given by $\alpha_1 \leq k_1$ and $\alpha_1 + \alpha_2 \leq k_2$, so that the space P is

$$\begin{aligned} P &= \operatorname{span}\left\{\xi_1^{j_1}\xi_2^{j_2} \mid j_1 \leq k_1, \quad j_1 + j_2 \leq k_2\right\}, \\ P &= \operatorname{span}\left\{\xi_1^{j_1}\xi_2^{j_2} \mid j_1 + j_2 \leq k\right\} = P_k, \quad (k = k_1 = k_2), \end{aligned} \tag{40}$$

where we have used P_k to denote the set of bivariate polynomials of total degree at most k.

4.3.3 Hexahedrons

We now move to three dimensions so $d = 3$, and taking $c = 0$ Duffy maps, again implying that $\boldsymbol{\xi} = \boldsymbol{\eta}$. Therefore, the space P on which the barycentric procedure is exact is $Q_{\boldsymbol{k}}$:

$$\begin{aligned} P &= \operatorname{span}\left\{\xi_1^{j_1}\xi_2^{j_2}\xi_3^{j_3} \mid \boldsymbol{j} \leq \boldsymbol{k}\right\} = Q_{\boldsymbol{k}}, \\ P &= \operatorname{span}\left\{\xi_1^{j_1}\xi_2^{j_2}\xi_3^{j_3} \mid j_q \leq k \ \forall \ q \in [3]\right\} = Q_{(k,k,k)}, \quad k = k_1 = k_2 = k_3. \end{aligned} \tag{41}$$

4.3.4 Prisms

As with hexahedral elements we have $d = 3$, but we now take $c = 1$, and a Duffy transformation collapse defined by $\boldsymbol{a} = 1, \boldsymbol{b} = 2$. Then, the $\boldsymbol{\eta} \leftrightarrow \boldsymbol{\xi}$ map is given by

$$\xi_1 = \frac{1}{2}(1+\eta_1)(1-\eta_2), \qquad \xi_2 = \eta_2, \qquad \xi_3 = \eta_3. \tag{42}$$

The constraints in the definition of A are given by $\alpha_1 \le k_1$ and $\alpha_1 + \alpha_2 \le k_2$, so that the space P is

$$\begin{aligned} P &= \operatorname{span}\left\{\boldsymbol{\xi}^{\boldsymbol{j}} \mid j_1 \le k_1, \;\; j_1 + j_2 \le k_2, \;\; j_3 \le k_3\right\}, \\ P &= \operatorname{span}\left\{\boldsymbol{\xi}^{\boldsymbol{j}} \mid j_1 + j_2 \le k, \;\; j_3 \le k\right\}, \quad (k = k_1 = k_2 = k_3). \end{aligned} \tag{43}$$

4.3.5 Tetrahedrons

Also with $d = 3$ and $c = 2$, a Duffy transformation collapses defined by $\boldsymbol{a} = (1, 2)$ and $\boldsymbol{b} = (2, 3)$, the $\boldsymbol{\eta} \leftrightarrow \boldsymbol{\xi}$ map is given by

$$\xi_1 = \frac{1}{2}(1+\eta_1)\left(1 - \frac{1}{2}(1+\eta_2)(1-\eta_3)\right), \quad \xi_2 = \frac{1}{2}(1+\eta_2)(1-\eta_3), \quad \xi_3 = \eta_3. \tag{44}$$

The constraints in the definition of A are given by $\alpha_1 \le k_1$ and $\alpha_1 + \alpha_2 \le k_2$, and $\alpha_1 + \alpha_2 + \alpha_3 \le k_3$ so that the space P is

$$\begin{aligned} P &= \operatorname{span}\left\{\boldsymbol{\xi}^{\boldsymbol{j}} \mid j_1 \le k_1, \;\; j_1 + j_2 \le k_2, \;\; j_1 + j_2 + j_3 \le k_3\right\}, \\ P &= \operatorname{span}\left\{\boldsymbol{\xi}^{\boldsymbol{j}} \mid j_1 + j_2 + j_3 \le k,\right\} = P_k, \quad (k = k_1 = k_2 = k_3). \end{aligned} \tag{45}$$

where we have used P_k to denote the set of trivariate polynomials of total degree at most k.

4.3.6 Pyramids

Finally, we again take $d = 3$ and $c = 2$, a Duffy transformation collapses defined by $\boldsymbol{a} = (1, 3)$ and $\boldsymbol{b} = (2, 3)$. Then, $\boldsymbol{\eta} \leftrightarrow \boldsymbol{\xi}$ map is given by

$$\xi_1 = \frac{1}{2}(1+\eta_1)(1-\eta_3), \qquad \xi_2 = \frac{1}{2}(1+\eta_2)(1-\eta_3), \qquad \xi_3 = \eta_3. \tag{46}$$

The constraints in the definition of A are given by $\alpha_1 \le k_1$ and $\alpha_1 \le k_2$, and $\alpha_1 + \alpha_2 + \alpha_3 \le k_3$ so that the space P is

$$\begin{aligned} P &= \operatorname{span}\left\{\boldsymbol{\xi}^{\boldsymbol{j}} \mid j_1 \le k_1, \;\; j_2 \le k_2, \;\; j_1 + j_2 + j_3 \le k_3\right\}, \\ P &= \operatorname{span}\left\{\boldsymbol{\xi}^{\boldsymbol{j}} \mid j_1 + j_2 + j_3 \le k,\right\} = P_k, \quad (k = k_1 = k_2 = k_3). \end{aligned} \tag{47}$$

4.4 Derivatives and Gradients

The results of Sect. 4.2 lead naturally to derivative evaluations. In particular, by defining the function $\hat{p} := p \circ D_{\boldsymbol{a},\boldsymbol{b}}$ in $\boldsymbol{\eta}$ space, and writing $p(\boldsymbol{\xi}) = \hat{p}(\boldsymbol{\eta}(\boldsymbol{\xi}))$, we can translate derivatives of p to those of $\hat{p}$, which can be efficiently evaluated using the results from previous sections.

Using the chain rule, the gradient of p can be written in terms of the gradient of $\hat{p}$,

$$\nabla_{\xi}\, p\,(\boldsymbol{\xi}) = \frac{\mathrm{D}\boldsymbol{\eta}}{\mathrm{D}\boldsymbol{\xi}} \nabla_{\boldsymbol{\eta}}\, \hat{p}\,(\boldsymbol{\eta})\,, \tag{48}$$

where $\nabla_{\boldsymbol{\eta}}$ is the standard d-variate gradient operator with respect to the Euclidean variables $\boldsymbol{\eta}$, and we have defined the $d \times d$ Jacobian matrix of the $\boldsymbol{\xi} \mapsto \boldsymbol{\eta}$ map,

$$\left(\frac{\mathrm{D}\boldsymbol{\eta}}{\mathrm{D}\boldsymbol{\xi}}\right)_{i,j} = \left(\frac{\mathrm{D}D^{-1}_{\boldsymbol{a},\boldsymbol{b}}(\boldsymbol{\xi})}{\mathrm{D}\boldsymbol{\xi}}\right)_{i,j} = \frac{\partial \eta_i}{\partial \xi_j}. \tag{49}$$

Note that the individual Duffy maps $D_{a,b}$ defined in (29) that collapse dimension a onto dimension b are invertible whenever $\eta_b \neq 1$, so that the Jacobian above is well defined away from these points. The formula above shows that since the gradient of $\hat{p}$ can be evaluated efficiently through the procedures in Sect. 4.4, so, too, can the gradient of p. In particular, this procedure exactly evaluates gradients (away from singularities of the Duffy transformation) if $p \in P(A)$ where $P(A)$ is given in (35).

Similarly, components of the Hessian of p can be evaluated as,

$$\frac{\partial^2 p}{\partial \xi_i \partial \xi_j} = \left(\frac{\partial^2 \boldsymbol{\eta}}{\partial \xi_i \partial \xi_j}\right)^T \nabla_{\boldsymbol{\eta}} \hat{p} + \left(\frac{\partial \boldsymbol{\eta}}{\partial \xi_i}\right)^T \boldsymbol{H}_{\boldsymbol{\eta}}(\hat{p}) \left(\frac{\partial \boldsymbol{\eta}}{\partial \xi_j}\right), \tag{50}$$

where $\frac{\partial^2 \boldsymbol{\eta}}{\partial \xi_i \xi_j} \in \mathbb{R}^d$ and $\frac{\partial \boldsymbol{\eta}}{\partial \xi_i} \in \mathbb{R}^d$ are componentwise derivatives, and $\boldsymbol{H}_{\boldsymbol{\eta}}(\hat{p})$ is the $d \times d$ Hessian of $\hat{p}$. Again, since the Hessian of $\widehat{p}$ can be efficiently evaluated through the procedures in Sect. 4.4, the Hessian of p also inherits this asypmtotic efficiency. This procedure again exactly evaluates Hessians (away from singularities of the Duffy transformation) if $p \in P(A)$. If $p \in P(A)$, higher-order derivatives of p may likewise be computed exactly from those of $\hat{p}$ using Faà di Bruno's formula with $\mathcal{O}\left(\prod_{j=1}^{d} k_j\right)$ complexity stemming from the multivarite barycentric procedures described earlier.

5 Algorithmic and Implementation Details

In this section, we present the implementation details of the barycentric Lagrange interpolation in terms of the data structures and algorithms involved. The implementation follows the high-level algorithms described in Sects. 3 and 4, but some details differ in service of computational routine optimization. The implementation of these concepts can be accessed in the open-source spectral/*hp* element library *Nektar*++ [2,15].

5.1 Algorithm

The foundation of the implementation is in the kernel that performs the barycentric interpolation itself as given in Eq. (1) – that is, it takes the coordinate of a single arbitrary point and the stored physical polynomial values at each quadrature point in the expansion and returns the interpolated value at the arbitrary point. This kernel has been templated to perform the interpolation only in a specific direction based on the integer template parameter DIR, and also to return the derivative value and second-derivative value by the reference parameter based on the boolean template parameters DERIV, and DERIV2. Templating here is defined in the sense of C++ templates; i.e. that these expressions are evaluated at compile time to reduce branching overheads and enable compiler inlining. For example, when DERIV and DERIV2 are not required, setting these template variables to false allows for performance gains by removing the if branch tests from the generated object code. The reasoning behind this unifying of the physical value evaluation and derivative interpolations is that we can make use of terms computed in the physical evaluation in the derivative interpolations saving repeat calculations (cf. (10) and (11)). An example kernel for physical, first- and second-derivative values is shown in Algorithm 1.

The next important method is the tensor-product function, which constructs the tensor line/square by calling the barycentric interpolation kernel on quadrature points, and is therefore dimension dependent and operates on the reference element in the appropriate form. The 1D version performs the barycentric interpolation directly on the provided point and returns the physical, first- and second- derivative value in the ξ_1 direction. In 2D and 3D, we chose to implement only the first-derivative to reduce overall complexity of the tensor-product method. The 2D version constructs an interpolation in the ξ_1 direction to give an intermediate step of physical values and derivative values at the expansion quadrature points in the same ξ_1 direction. The quadrature derivative values in the ξ_1 direction can then be evaluated in the ξ_2 direction to produce the single derivative value in the ξ_1 direction at the point provided. Likewise, the quadrature physical values in the ξ_1 direction can then be evaluated in the ξ_2 direction with the derivative output enabled to return both the single derivative value in ξ_2 direction and the physical value at the provided point. The tensor product in 3D is similar, except we now also consider the ξ_3 direction and so our intermediary steps consist of constructing the tensor square. Structuring it in this manner allows for a minimum number of calls to the barycentric interpolation kernel. An example tensor product function in 2D is shown in Algorithm 2.

As this tensor-product function operates on the reference element that in 2D is a quadrilateral, and in 3D a hexahedron, additionally, it can be extended to non-reference shape types by collapsing coordinates and performing the correct quadrature point mapping as described in Sect. 4.1. This is achieved by overriding the existing reference element interpolation function, which evaluates the expansion at a single (arbitrary) point of the domain to also give it the capabilities to evaluate the derivative in each direction as needed. This function is a wrapper around a virtual function that is defined for each shape type and therefore allows for the shape dependent coordinate collapsing. Example structures of these functions for a triangular shape type are shown in Algorithms 3 and 4.

5.2 Complexity Analysis

Given a function $p(\eta_{\mathtt{DIR}})$ evaluated at $k+1$ quadrature points $Q = \{z_0, z_1, \cdots, z_k\}_{\mathtt{DIR}}$, we perform the following steps to find the interpolated values of $p(\eta)$, and $\frac{\partial p}{\partial \eta_{\mathtt{DIR}}}$ at a given point $\eta \notin Q$:

(a) Calculate and store the weights $\{w_j\}, \forall j = 0, 1, \cdots k$ as per (2), which requires storage of size $k+1$. The number of flops for this operation is $(k+1)^2 + 1$, and the complexity is $O(k^2)$, which is a one-time setup cost.

Algorithm 1 Example kernel for the Barycentric interpolation of a single point to provide physical and first-derivative values dependent on the template parameters provided. The $\langle\ldots\rangle$ indicates template arguments, $(\ldots)$ indicates normal arguments, and $\bullet$ is a matrix-vector multiplication operation.

```
procedure BaryEvaluate ⟨DIR, DERIV = FALSE, DERIV2 = FALSE⟩ $(\eta, p)$
    $A = 0, B = 0, C = 0, D = 0, E = 0, F = 0$
    for each quadrature point, $z_{\mathrm{DIR}}$ do
        $x = z_{\mathrm{DIR}} - \eta$
        if $x = 0$ then
            $p_{\eta_{\mathrm{DIR}}} = p_{z_{\mathrm{DIR}}}$
            if DERIV2 then
                $\frac{dp}{d\eta_{\mathrm{DIR}}} = \boldsymbol{D}_{z_{\mathrm{DIR}}} \bullet p$        ▷ Use the precomputed derivative matrix, $\boldsymbol{D}_z$
                $\frac{d^2p}{d\eta^2_{\mathrm{DIR}}} = \boldsymbol{D}^2_{z_{\mathrm{DIR}}} \bullet p$        ▷ Use the precomputed 2nd derivative matrix, $\boldsymbol{D}^2_z$
                $\boldsymbol{out} \leftarrow p_{\eta_{\mathrm{DIR}}}, \frac{dp}{d\eta_{\mathrm{DIR}}}, \frac{d^2p}{d\eta^2_{\mathrm{DIR}}}$
            else if DERIV then
                $\frac{dp}{d\eta_{\mathrm{DIR}}} = \boldsymbol{D}_{z_{\mathrm{DIR}}} \bullet p$        ▷ Use the precomputed derivative matrix, $\boldsymbol{D}_z$
                $\boldsymbol{out} \leftarrow p_{\eta_{\mathrm{DIR}}}, \frac{dp}{d\eta_{\mathrm{DIR}}}$
            else
                $\boldsymbol{out} \leftarrow p_{\eta_{\mathrm{DIR}}}$
            end if
        end if
        $t_1 = w_{z_{\mathrm{DIR}}}/x$
        $A = A + t_1 * p_{z_{\mathrm{DIR}}}$
        $F = F + t_1$
        if DERIV or DERIV2 then
            $t_2 = t_1/x$
            $B = B + t_2 * p_{z_{\mathrm{DIR}}}$
            $C = C + t_2$
            if DERIV2 then
                $t_3 = t_2/x$
                $D = D + t_3 * p_{z_{\mathrm{DIR}}}$
                $E = C + t_3$
            end if
        end if
    end for
    $p_{\eta_{\mathrm{DIR}}} = A/F$
    if DERIV or DERIV2 then
        $FF = F * F$
        $AC = A * C$
        $\frac{dp}{d\eta_{\mathrm{DIR}}} = (B * F - AC)/FF$
        if DERIV2 then
            $\frac{d^2p}{d\eta^2_{\mathrm{DIR}}} = (2 * D)/F - (2 * E * A)/FF - (2 * B * C)/FF + (2 * C * AC)/(FF * F)$
            $\boldsymbol{out} \leftarrow p_{\eta_{\mathrm{DIR}}}, \frac{dp}{d\eta_{\mathrm{DIR}}}, \frac{d^2p}{d\eta^2_{\mathrm{DIR}}}$
        end if
        $\boldsymbol{out} \leftarrow p_{\eta_{\mathrm{DIR}}}, \frac{dp}{d\eta_{\mathrm{DIR}}}$
    else
        $\boldsymbol{out} \leftarrow p_{\eta_{\mathrm{DIR}}}$
    end if
end procedure
```

Algorithm 2 Sum factorization using the `BaryEvaluate` function for 2D shape types to give physical and first-derivative values. The variables $phys0$ and $deriv0$ refer to arrays which are populated with the physical and derivative values respectively, taken at each line of points denoted by index i in the ξ_1 direction, as described in Sect. 5.1.

```
procedure BaryTensorDeriv($\eta$, $p$)
    for each quadrature point in $\eta_2$ direction, $x_2$ do
        $phys0[i], deriv0[i] \leftarrow$ BaryEvaluate$\langle 0, true\rangle(\eta_1, p_{(i \times k_1)})$
    end for
    $\frac{dp}{d\eta_1} \leftarrow$ BaryEvaluate$\langle 1\rangle(\eta_2, deriv0[0])$
    $p_\eta, \frac{dp}{d\eta_2} \leftarrow$ BaryEvaluate$\langle 1, true\rangle(\eta_2, phys0[0])$
    out $\leftarrow p_\eta, \frac{dp}{d\xi_1}, \frac{dp}{d\xi_2}$
end procedure
```

Algorithm 3 Coordinate collapsing for a triangle.

```
procedure CollapseCoords($\xi$)
    if $\xi_y = 1$ then
        $\eta_1 = -1$
        $\eta_2 = 1$
    else
        $\eta_1 = 2 * (1 + \xi_1)/(1 - \xi_2) - 1$
        $\eta_2 = \xi_2$
    end if
    out $\leftarrow \eta$
end procedure
```

Algorithm 4 Overview of the Barycentric solution and first-derivative evaluation for a triangle.

```
procedure PhysEvaluate($\xi$, $p$)
    $\eta \leftarrow$CollapseCoords($\xi$)
    $p_\eta, \frac{dp}{d\eta_1}, \frac{dp}{d\eta_2} \leftarrow$ BaryTensorDeriv($\eta$, $p$)
    set up geometric factor for x derivative $G_1 = 2/(1 - \eta_2)$
    set up geometric factor for y derivative $G_2 = G_1 * (\eta_1 + 1)/2$
    $\frac{dp}{d\xi_1} = \frac{dp}{d\eta_1} * G_1$
    $\frac{dp}{d\xi_2} = \frac{dp}{d\eta_2} + G_2 * \frac{dp}{d\eta_1}$
    out $\leftarrow p_\xi, \frac{dp}{d\xi_1}, \frac{dp}{d\xi_2}$
end procedure
```

(b) Calculate $p(\eta)$ using step 32 of Algorithm 1, for which we use the pre-computed weights w from previous step and calculate the terms A and F. The storage requirement for each of these terms is of size $k + 1$, and the number of flops required to calculate them is $3(k + 1)$ and $2(k + 1)$, respectively. (If we reuse the term t_1 from step 18, calculating F needs only $k + 1$ flops). Thus, the total complexity of finding $p(\eta)$ using the barycentric method is $O(k)$, which is consistent with the evaluation in [1].

(c) Calculate $\frac{\partial}{\partial \eta_{\mathtt{DIR}}} p(\eta)$ as shown in step 36 of Algorithm 1 which uses the precomputed weights w and the terms A and F from the previous step. Additional terms B and C are evaluated as per steps 23 and 24 of Algorithm 1. The storage requirement for these terms is of size $k + 1$ each. The calculation of term B requires $4(k + 1)$ flops (or $3(k + 1)$ using precomputed t_1). Similarly, calculating term C requires $2(k + 1)$ flops (or $k + 1$

if we consider precomputed t_2). Therefore, the total complexity of evaluating $\frac{\partial}{\partial \eta_{\mathtt{DIR}}} p(\eta)$ is $O(k)$.

(d) Applying a similar analysis for $\frac{d^2}{d\eta^2_{\mathtt{DIR}}} p(\eta)$, we need to evaluate additional terms D and E as shown in steps 27 and 28 of Algorithm 1. We need additional storage of size $k + 1$ for each of these terms. The computational complexity for both D and E is $O(k)$.

Note that the analysis presented above is independent of dimensions ($\mathtt{DIR}$). For higher dimensions, we follow the same procedure in each individual direction. For example, in 2D we require evaluation of p for $\mathtt{DIR} = 1$ and $\mathtt{DIR} = 2$. Therefore, when $\mathtt{DIR} = 2$, the algorithm takes twice the amount of calculations as 1D. Thus, the complexity of the evaluation is still $O(k)$. Similarly, for the first-derivative, we need the individual evaluations $\partial p/\partial \eta_0$ and $\partial p/\partial \eta_1$. Therefore, the computational complexity of the derivative evaluation is $O(k)$. By extension, the computational complexity for the second-derivative is also $O(k)$.

6 Evaluation and Comparison

6.1 Baseline Evaluations

To investigate how this implementation of the barycentric interpolation affects the efficiency of evaluation for physical and derivative values, a number of tests were run across various cases. The barycentric interpolation method has been implemented within the *Nektar++* spectral/*hp* element framework [2,15] in a discontinuous Galerkin (DG) setting and is compared with two variants of the already existing standard Lagrange interpolation method, the first where the interpolation matrix is recalculated every iteration, and the second where the interpolation matrix is stored across iterations, mimicking a scenario involving history points. All test cases below were carried out on a single core of a dual-socket Intel Xeon Gold 5120 system, equipped with 256GB of RAM, with the solver pinned to a specific core in order to reduce the influence of kernel core and socket reassignment mid-process.

6.1.1 Construction of the Baseline Tests

These baseline tests are constructed for the desired elemental shape using the hierarchical modified basis of Karniadakis & Sherwin [10] of order P with tensor products of $P + 2$ points in each direction. We make use of Gauss-Lobatto-Legendre points in noncollapsed directions and Gauss-Radau points in collapsed directions to avoid evaluation at singularities. The physical values at these points are provided by the polynomial $p(\boldsymbol{\xi}) = \xi_1^2 + \xi_2^2 - \xi_3^2$, which also allows for an analytical solution at any $\boldsymbol{\xi}$ for the physical and derivative values in each direction. The physical and derivative values are sampled on the constructed shape on a collocation grid that is again constructed as GLL/GLR points, like the quadrature rule. However, we use a fixed collocation grid size while varying order P, so that we ensure that the collocation grid is distinct from the quadrature rule in most cases. To ensure the same number of points is being sampled for all shape dimensions, we choose to use 64 total points because of the symmetry so that in 1D, it is 64^1, 2D it is 8^2, and 3D it is 4^3. This creates some special considerations when the collocation grid matches exactly with the quadrature points used within the shape, which we discuss in the relevant sections below. We average the timings, in 1D from 10^6 evaluations, and in 2D/3D from 10^5 evaluations, to ensure results are not affected by system noise or other external factors. The tests are performed for a range

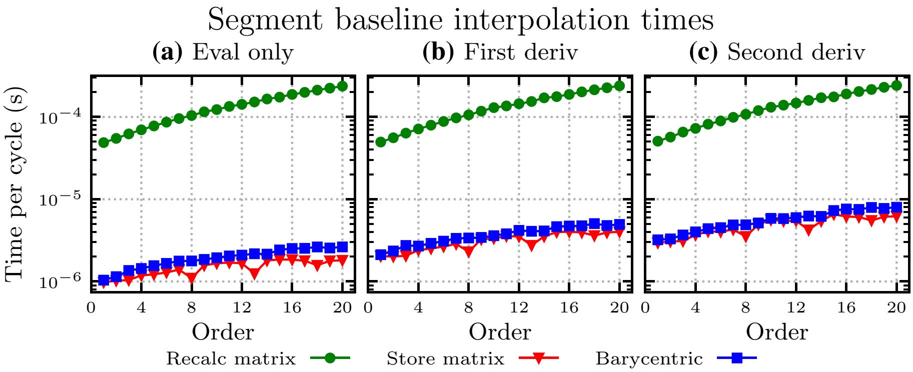

Fig. 2 Baseline interpolation timings for a segment. **a** Only physical values, **b** Physical and first derivative values, **c** Physical and first-, and second-derivative values

of basis orders from 2 to 20. In 1D, we calculate the physical, first and second derivative values, whereas in 2D and 3D, we calculate only the physical and first derivative values.

6.1.2 1D barycentric Interpolation and Derivatives

Figure 2 shows that recalculating the interpolation matrix every cycle is the notably slower of the three methods, whereas the barycentric interpolation and stored interpolation matrix method are closer in performance with the barycentric interpolation being on average across the orders 33% slower for the solution evaluation only, 20% slower when including first derivatives, and 18% slower when including second derivatives. However, the barycentric interpolation wins in terms of the storage complexity. This is because the former requires $\mathcal{O}(k)$ storage to store the weights w_j, where k and z_j are given and fixed. On the other hand, the stored interpolation matrix has the best case space complexity of $\mathcal{O}(k^2)$. An interesting feature present is the minor reductions in interpolation time for the stored matrix method at order 3, 8, 13, and 18. These basis orders correspond to the number of quadrature points being a multiple of five, which we theorize is the line cache size of the CPU being used. A match of this cache size with the quadrature point array sizes in our implementation will result in memory optimizations for the interpolation matrix multiplications. This phenomena will also be present for the recalculated matrix method, however, the result of optimization in this case is not visible on the graph due to the larger time scale. It can be seen that the baseline computational cost increases moving from interpolating the physical values only (Fig. 2a) to also including the first derivatives (Fig. 2b), and then the second derivatives (Fig. 2c) for all methods.

6.1.3 Extension to Traditional Tensor-Product Expansions

For the traditional tensor-product expansions, we now consider a quadrilateral element in 2D and a hexahedron in 3D. Figure 3 shows the results for the quadrilateral element. Generally results are similar to that for the segment. An obvious unique feature is the spike in the recalculated matrix timing result at order 6. The spike corresponds to 8 quadrature points in each direction, which is the same as the number we are sampling on, and therefore the points are collocated. Consequently, the routine in which the interpolation matrix is constructed

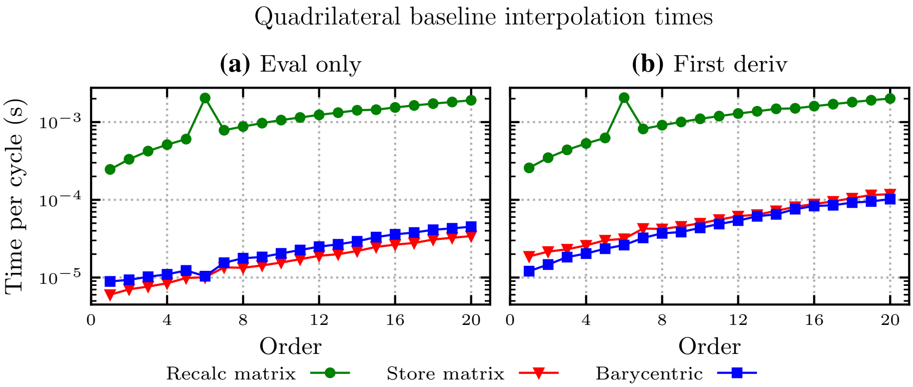

Fig. 3 Baseline interpolation timings for a quadrilateral. **a** Only physical values, **b** Physical and first-derivative values

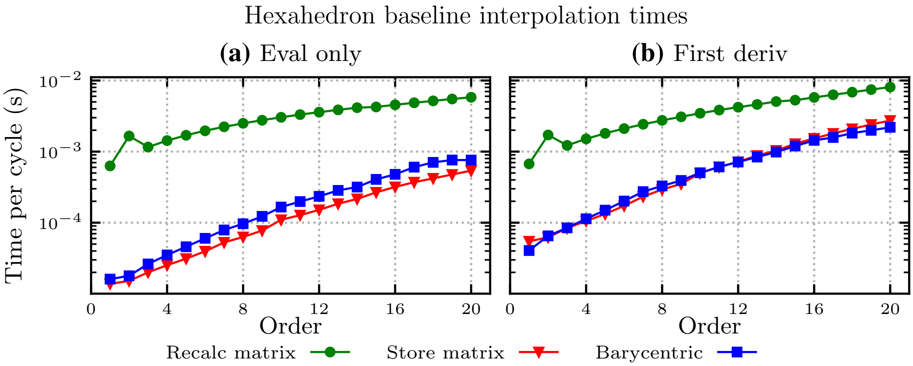

Fig. 4 Baseline interpolation timings for a hexahedron. **a** Only physical values, **b** Physical and first derivative values

has to handle this collocation, which results in the increased cost. We can also see the the barycentric interpolation method handling this collocation, resulting in a speed-up compared to neighboring orders, evident in Fig. 3a. On average the barycentric interpolation method is 30% slower across the orders for the solution evaluation only when compared to the stored interpolation matrix method. Figure 3b including the first derivative evaluations shows that the barycentric interpolation method is on average 15% faster across the orders than the stored matrix variant of the Lagrangian method indicating the computational cost savings from unifying the derivative call.

The same collocation trend is present in the hexahedral element, shown in Fig. 4 with the spike now present at order 2, which corresponds with the 4 quadrature points in each direction. The trends are similar again to the 1D and 2D results. On average across the orders for the the solution evaluation only the barycentric interpolation method is 48% slower than the stored matrix interpolation method. The most notable difference compared to the 2D results is in the first derivative timings (Fig. 4b), which when disregarding order 2 demonstrates the barycentric interpolation method is on average 10% slower at orders ≤ 11, while at orders > 11 it is on average 9% faster.

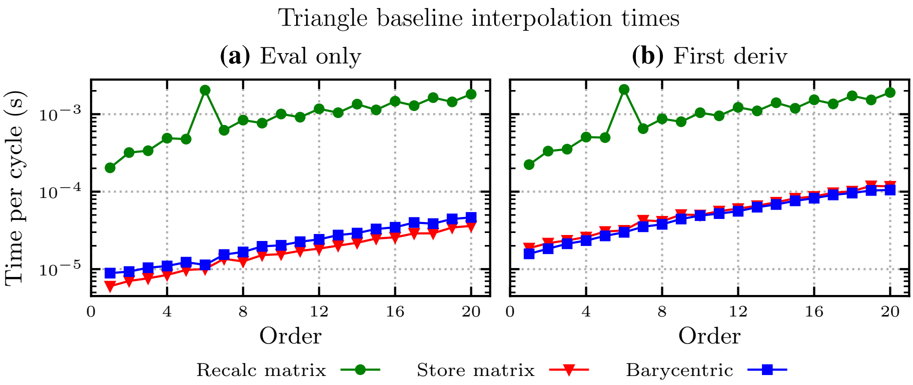

Fig. 5 Baseline interpolation timings for a triangle: **a** Only physical values and **b** Physical and first-derivative values

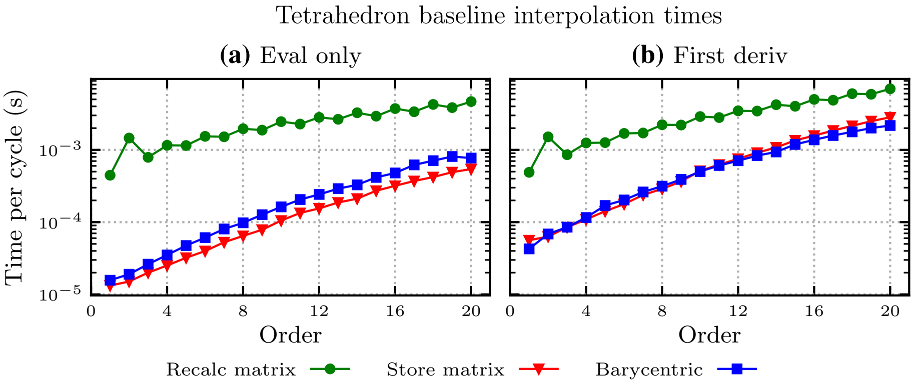

Fig. 6 Baseline interpolation timings for a tetrahedron: **a** Only physical values and **b** Physical and first-derivative values

6.1.4 Extension to General Expansions

We now compare the most complicated of the available shape types in two and three dimensions, the triangle and the tetrahedron, which require the use of Duffy transformations. The results shown in Figs. 5 and 6 align closely with their tensor-product expansion counterparts, the quadrilateral and hexahedron, respectively. A unique feature can now be seen in the recalculated matrix interpolation method that appears to show a odd/even cyclical trend. We believe this is again due to collocated points, this time as a consequence of the collapsing of the element and the routine used to calculate the interpolation matrix making use of a floor function.

6.1.5 Speed-Up Factor

To further compare the methods, Figure 7 shows the speed-up factor when going from the recalculated matrix variant of the Lagrangian interpolation method to the barycentric interpolation method for segments, quadrilaterals, and hexahedrons. This shows that in 1D as the order increases the speed-up increases, for 2D it stays approximately the same, and for 3D it

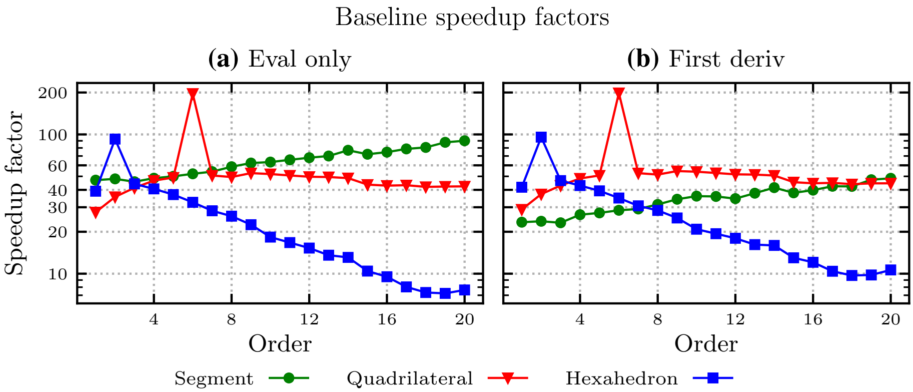

Fig. 7 Speed-up factors calculated for the segment, quadrilateral, and hexahedron: **a** Only physical values and **b** Physical and first-derivative values

decreases. We can see that including the first-derivatives (Fig. 7b) in 1D causes the speedup factor to reduce when compared with the evaluation only version (Fig. 7). In 2D, the speed-up factor remains approximately consistent between the two versions, and in 3D it increases. A minimum speedup factor of approximately 7 is observed across all tests occurring in hexahedrons greater than order 17 when calculating the solution evaluation only.

6.2 Real-World Usage Example

To investigate the performance of the barycentric interpolation method in a less artificial setting, we now consider a real world problem containing a nonconformal interface, again posed in a DG setting within the Nektar++ spectral/*hp* element framework. In general we can imagine two scenarios: the first in which the nonconformal interface is fixed in time, and the second where the interface changes at each timestep to account for e.g. a grid rotation or translation. We investigate both settings in this section.

6.2.1 Handling the Nonconformal Interface

To handle the transfer of information across a nonconformal interface we adopt a point-to-point interpolation approach as outlined in [12], which involves minimizing an objective function to find an arbitrary point on a curved element edge utilizing the inverse of a parametric mapping to the reference element. In our implementation, this minimization problem is solved via a gradient-descent method utilizing a quasi-Newton search direction and backtracking line search that makes use of repeated calls to determine the physical, first- and second-derivative values within the loop. Once calculated and for a stationary interface the location of this arbitrary point in the reference element can be cached; however, to mimic a moving interface, where the minimization routine must be run every timestep, we disable this caching in order to also evaluate the performance impact of the new barycentric interpolation method. This is the equivalent of the comparison to the first Lagrange interpolation method discussed above, where the interpolation matrix is recalculated every iteration.

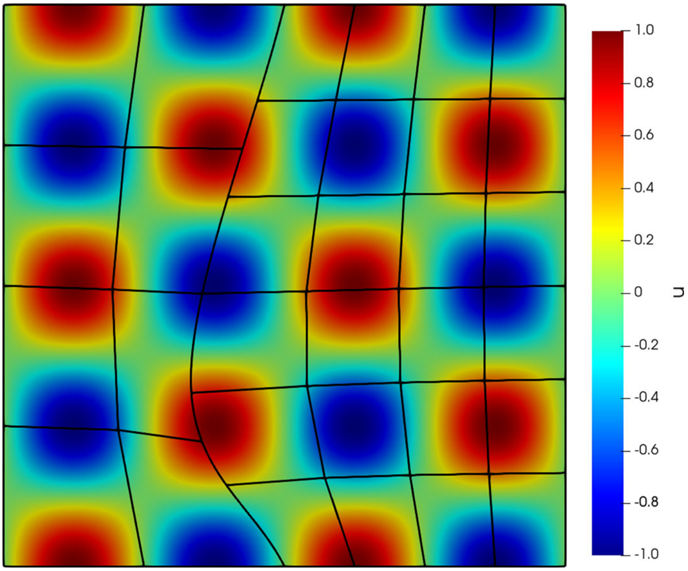

Fig. 8 The nonconformal mesh for the real world usage example with the initial projection of the u field overlaid

Table 2 Timings for the real world case using the Barycentric and Lagrangian method

	Avg. cost per timestep (s)		
Method	Cached	Noncached	Minimization
Lagrangian	0.00162059	0.021594	0.01997341
Barycentric	0.00163341	0.00483588	0.00320247

6.2.2 Test Case

We select a standard linear transport equation $u_t + \nabla \cdot \boldsymbol{F}(u) = 0$ within a domain $\Omega = [-1, 1]^2$, so that $\boldsymbol{F}(u) = \boldsymbol{v}u$ for a constant velocity $\boldsymbol{v} = (1, 0)$, and an initial condition that is nonpolynomial, so that $\boldsymbol{u}(\boldsymbol{x}, 0) = \sin(2\pi x)\cos(2\pi y)$. The domain consists of a single nonconformal interface with unstructured quadrilateral subdomains on either side, as visualized in Fig. 8 together with the initial condition for u. This means that the interpolation is being performed on the trace edges of the elements located at the nonconformal interface, which in this 2D example are segments. A polynomial order of $P = 8$ is considered, and we select $Q = P + 2 = 10$ quadrature points in each coordinate direction. We select a timestep size of $\Delta t = 10^{-3}$ and time for 10 cycles (i.e., $t = 10$), which is the equivalent of 10^4 timesteps.

We initially obtain a baseline time for both interpolation methods with the cache enabled. The cache is then disabled and both methods run again, allowing us to compare the cached (static) version with the non-cached (moving) version as a demonstration of the high computational cost incurred by calling this minimization routine every timestep. The results for the cached and noncached versions are shown in Table 2. This demonstrates that with the cache enabled, the timings for both methods are practically identical because the minimization occurs only in the first timestep which incurs a negligible cost over this timescale. However, the non-cached results (where the minimization procedure is run every timestep) shows a slowdown of around 13× for the Lagrangian method, but only 3× for the barycentric method when compared with the cached results. We can then calculate the performance impact of these methods only on the minimization routine, which shows the routine using the barycentric method as around 6× faster than the equivalent routine using the Lagrangian method. This is a significant speed-up, as the minimization routine accounts for a large proportion of the total computational time: for the Lagrangian method, this routine occupies 92% of total time whereas using the barycentric approach reduces this to 66%. The speed-up is realized in the total time taken for all 10^4 timesteps being reduced from 216s to 48s.

7 Conclusions

In the context of spectral/*hp* and high-order finite elements, solution expansion evaluation at arbitrary points in the domain has been a core capability needed for postprocessing operations such as visualization (streamlines/streaklines and isosurfaces) as well for interfacing methods such as mortaring. The process of evaluation of a high-order expansion at an arbitrary point in the domain consists of two parts: determining in which particular element the point lies, and evaluating the expansion within that element. This work focuses on efficient solution expansion evaluation at arbitrary points within an element. We expand barycentric interpolation techniques developed on an interval to 2D (triangles and quadrilaterals) and 3D (tetrahedra, prisms, pyramids, and hexahedra) spectral/*hp* element methods. We provided efficient algorithms for their implementations, and demonstrate their effectiveness using the spectral/*hp* element library *Nektar++*. The barycentric method shows a minimum speedup factor of 7 when compared with the non-cached interpolation matrix version of the Lagrangian method across all tests demonstrating a good performance uplift, culminating in an approximately 6× computational time speedup for the real-world example of advection across a nonconformal interface. In the artificial tests the barycentric method exhibits slightly worse performance than the stored interpolation matrix version of the Lagrangian method when evaluating purely physical values, with slowdowns of between 10% and 50% across all orders dependant on element type. However if first derivatives are also required the barycentric method can outperform the stored interpolation matrix method by up to 35%.

Funding The first and fifth authors acknowledge support from the EPSRC Platform Grant PRISM under grant EP/R029423/1 and the ELEMENT project under grant EP/V001345/1. The second and fourth authors acknowledge support from ARO W911NF-15-1-0222 (Program Manager Dr. Mike Coyle). The third author acknowledges support from NSF DMS-1848508.

Declarations

Conflict of interest The authors declare no conflicts of interest.

Code Availability All code is available in the *Nektar++* repository at https://gitlab.nektar.info.

References

1. Berrut, J.P., Trefethen, L.N.: Barycentric lagrange interpolation. SIAM Rev. **46**, 501–517 (2004)
2. Cantwell, C.D., Moxey, D., Comerford, A., Bolis, A., Rocco, G., Mengaldo, G., De Grazia, D., Yakovlev, S., Lombard, J.E., Ekelschot, D., Jordi, B., Xu, H., Mohamied, Y., Eskilsson, C., Nelson, B., Vos, P., Biotto, C., Kirby, R.M., Sherwin, S.J.: Nektar++: An open-source spectral/hp element framework. Comput. Phys. Commun. **192**, 205–219 (2015). https://doi.org/10.1016/j.cpc.2015.02.008
3. Chooi, K., Comerford, A., Sherwin, S., Weinberg, P.: Intimal and medial contributions to the hydraulic resistance of the arterial wall at different pressures: a combined computational and experimental study. J. R. Soc. Interface **13**(119), 20160234 (2016)
4. Clenshaw, C.W.: A note on the summation of Chebyshev series. Math. Comput. **9**(51), 118–120 (1955)
5. Deville, M., Mund, E., Fischer, P.: High Order Methods for Incompressible Fluid Flow. Cambridge University Press, Cambridge (2002)
6. Hesthaven, J.S., Warburton, T.: Nodal discontinuous Galerkin methods: algorithms. Analysis and Applications. Springer, New York, NY (2007)
7. H.T. Huynh, Z.W., Vincent, P.: High-order methods for computational fluid dynamics: a brief review of compact differential formulations on unstructured grids. Computers & Fluids **98**, 209–220 (2014)
8. Jallepalli, A., Kirby, R.M.: Efficient algorithms for the line-SIAC filter. J. Sci. Comput. **80**, 743–761 (2019)
9. Jallepalli, A., Levine, J.A., Kirby, R.M.: The effect of data transformation methodologies on the topological analysis of high-order FEM solutions. IEEE Trans. Vis. Comput. Graph. **26**, 162–172 (2020)
10. Karniadakis, G., Sherwin, S.: Spectral/hp Element Methods for Computational Fluid Dynamics, 2nd edn. Oxford University Press, Oxford (2005)
11. Kelly, M.: An introduction to trajectory optimization: How to do your own direct collocation. SIAM Rev. **59**, 849–904 (2017)
12. Laughton, E., Tabor, G., Moxey, D.: A comparison of interpolation techniques for non-conformal high-order discontinuous Galerkin methods. Comput. Methods Appl. Mech. Eng. **381**, 381–113820 (2021). https://doi.org/10.1016/j.cma.2021.113820
13. Lombard, J.E.W., Moxey, D., Sherwin, S.J., Hoessler, J.F.A., Dhandapani, S., Taylor, M.J.: Implicit large-eddy simulation of a wingtip vortex. AIAA J. **54**(2), 506–518 (2016)
14. Moxey, D., Amici, R., Kirby, R.M.: Efficient matrix-free high-order finite element evaluation for simplicial elements. SIAM J. Sci. Comput. **43**, C97-123 (2020)
15. Moxey, D., Cantwell, C.D., Bao, Y., Cassinelli, A., Castiglioni, G., Chun, S., Juda, E., Kazemi, E., Lackhove, K., Marcon, J., Mengaldo, G., Serson, D., Turner, M., Xu, H., Peiro, J., Kirby, R.M., Sherwin, S.J.: Nektar++: enhancing the capability and application of high-fidelity spectral/hp element methods. Comput. Phys. Commun. **249**, 107110 (2020)
16. Sirisup, S., Karniadakis, G.E., Xiu, D., Kevrekidis, I.G.: Equation-free/Galerkin-free POD-assisted computation of incompressible flows. J. Comput. Phys. **207**(2), 568–587 (2005)
17. Steffen, M., Curtis, S., Kirby, R.M., Ryan, J.K.: Investigation of smoothness-increasing accuracy-conserving filters for improving streamline integration through discontinuous fields. IEEE Trans. Vis. Comput. Graph. **14**(3), 680–692 (2008)