http://www.aimspress.com/journal/mine

Mathematics in Engineering, 6(1): 192–220.
DOI:10.3934/mine.2024009
Received: 30 July 2023
Revised: 15 December 2023
Accepted: 14 February 2024
Published: 05 March 2024

Research article

Quadrature-free polytopic discontinuous Galerkin methods for transport problems[†]

Thomas J. Radley, Paul Houston[*] and Matthew E. Hubbard

School of Mathematical Sciences, University of Nottingham, University Park, Nottingham NG7 2RD, UK

[†] **This contribution is part of the Special Issue:** Advancements in Polytopal Element Methods
Guest Editors: Michele Botti; Franco Dassi; Lorenzo Mascotto; Ilario Mazzieri
Link: `www.aimspress.com/mine/article/6538/special-articles`

* **Correspondence:** Email: paul.houston@nottingham.ac.uk; Tel: +441158467468.

Abstract: In this article we consider the application of Euler's homogeneous function theorem together with Stokes' theorem to exactly integrate families of polynomial spaces over general polygonal and polyhedral (polytopic) domains in two and three dimensions, respectively. This approach allows for the integrals to be evaluated based on only computing the values of the integrand and its derivatives at the vertices of the polytopic domain, without the need to construct a sub-tessellation of the underlying domain of interest. Here, we present a detailed analysis of the computational complexity of the proposed algorithm and show that this depends on three key factors: the ambient dimension of the underlying polytopic domain; the size of the requested polynomial space to be integrated; and the size of a directed graph related to the polytopic domain. This general approach is then employed to compute the volume integrals arising within the discontinuous Galerkin finite element approximation of the linear transport equation. Numerical experiments are presented which highlight the efficiency of the proposed algorithm when compared to standard quadrature approaches defined on a sub-tessellation of the polytopic elements.

Keywords: polytopic elements; numerical integration; discontinuous Galerkin methods

1. Introduction

Over the past 10–15 years there has been increasing widespread interest in the development of numerical methods for the approximation of partial differential equations (PDEs) based on employing general element shapes such as polygons in 2D and polyhedra in 3D (which we collectively refer to as polytopic elements), cf. [1, 3, 5, 6, 9], for example, and the references cited therein. Exploiting such flexible computational meshes is highly attractive for a number of key reasons: Complicated geometries may be meshed by employing relatively coarse meshes, without the need to simplify the boundary or other features within the given domain; moving meshes and/or overlapping meshes arising, for example, in applications such as fluid structure interaction can easily be accommodated; hanging nodes are treated in a very simple and natural manner; finally, multi-level solvers, such as domain decomposition methods and multigrid can easily be implemented employing embedded polytopic meshes.

However, a major bottleneck in the exploitation of general polytopic elements within, for example, finite element methods is the construction of suitable element quadratures needed for the assembly of the underlying matrix system. With this in mind several approaches have been proposed within the literature. The simplest approach is to construct a sub-tessellation of a given polytope into standard element shapes, for example, simplices and/or tensor-product elements (quadrilaterals in 2D and hexahedra in 3D) and employ known quadrature rules on these sub-elements. We remark that when the underlying polytopic mesh is constructed from the agglomeration of a given fine mesh consisting of standard element shapes, then the sub-tessellation need not be computed, as it will already be provided. The major problem with this approach is that the number of quadrature points can be extremely large, depending on the cardinality of the sub-tessellation and the required accuracy. Of course, quadrature is naturally highly parallisable, and hence such an implementation can be accelerated, cf. [10] for example, who employed a GPU approach. Alternatively, one may for example, use this initial quadrature and attempt to optimise it by successively removing points until a minimal number is attained; here, we mention the moment fitting approaches, cf. [19–21, 24, 27], for example, and the references cited therein.

In this article we pursue an alternative approach based on the integration of homogeneous functions. This idea was first developed in the articles [7, 18], cf. also [8]. Here, the essential idea is to employ Euler's homogeneous function theorem, together with Stokes' theorem, which allows for the integral of a homogeneous function over a given polytope to be written as a boundary integral. Recursively applying this approach allows for the integral to be exactly evaluated based on only computing values of the integrand and its derivatives at the vertices of the polytopic domain, and hence leads to an exact quadrature rule whose quadrature points are the vertices of the polytope. In our recent paper [2] we considered the application of this algorithm within the discontinuous Galerkin finite element (DGFEM) approximation of the Poisson problem. In this article, we extend this work by presenting a detailed complexity analysis of both a quadrature based algorithm and the proposed quadrature-free approach. Here, the primary focus is to consider the number of flops required to integrate an entire space of polynomials up to a given fixed degree, which is typically required within a finite element implementation. In particular, we show that the computational complexity of the proposed algorithm depends on three key factors: the ambient dimension of the underlying polytopic domain; the size of the requested polynomial space to be integrated; and the size of a directed graph related to the polytopic domain. By a careful selection of the so-called local origins needed on each of the polytope's lower dimensional facets, which is needed in the specification of the quadrature-free algorithm, we are able to considerably optimise the number of required flops within this approach by reducing the size of the associated directed graph. To demonstrate the application of this optimised quadrature-free algorithm, we utilise this within a DGFEM approximation of the linear transport equation, though we stress that this approach can naturally be employed for the numerical approximation of a wide range of PDE problems. We point out that the specific application we have in mind is the numerical approximation of the linear Boltzmann transport equation. This is a high-dimensional integro-differential equation employed within applications including neutron and radiation transport. In our recent article [13], we illustrate that by employing a judicious choice of local basis functions and quadratures within the angular and energy domains, the DGFEM approximation of the underlying problem may be computed by simply solving a sequence of linear transport equations in space corresponding to each angular quadrature point (which defines a constant advection direction) and each energy quadrature point. In typical 3D applications utilising either source iteration, or a source iteration preconditioner within a Krylov space solver, means that an extremely large number of these transport solves must be computed. With that in mind, efficiency is of paramount concern, and hence the need to develop optimised quadrature-free implementations of the linear transport equation. Numerical experiments presented here highlight the gains in efficiency with our proposed optimised quadrature-free algorithm in comparison with standard sub-tessellation based quadrature. In particular, we study the dependence of both algorithms on the shape of the underlying polytope.

The outline of this article is as follows. In Section 2 we recall the quadrature-free integration technique introduced in [7] and its application to the integration of families of monomial functions over general d-dimensional polytopes as in [2]. The time and space (memory) complexities of this algorithm in the case where the integration domain is polygonal or polyhedral are analysed. In Section 3 we introduce the DGFEM approximation of a linear first-order transport equation on general polytopic meshes. In Section 4 we compare the time complexities associated with the assembly of the local DGFEM element matrices via quadrature based and quadrature-free based approaches, where we exploit a known decomposition of the integrands into a linear combination of monomials. For simplicity of presentation, we do not consider the application of the quadrature-free approach for the evaluation of the face terms arising within the DGFEM, but instead refer to our previous paper [2] where this issue is discussed. In particular, we recall that the monomial coefficients of the underlying basis restricted to a given face must be computed *online*, i.e., for every face individually. As a consequence, the numerical experiments presented in [2] indicate that when the faces consist of standard shapes, for example, simplices or tensor-product shapes (when a sub-mesh is not required to compute an appropriate quadrature) the speed-up facilitated by employing the quadrature-free approach is relatively modest, compared to the cost of exploiting a standard quadrature rule. Though, again, we stress that when the faces consist of arbitrary planar polytopes the application of a sub-mesh quadrature approach may become very expensive. Several numerical experiments are performed in Section 5 in order to demonstrate the accelerated assembly time of the quadrature-free based approach. Finally, in Section 6 we summarise the work undertaken in this article and discuss future extensions.

2. Quadrature-free integration of monomials

In this section we review the procedure for the numerical integration of homogeneous functions over a polytopic domain as introduced by Lasserre [17, 18] for convex polytopes and extended by Chin et al. in [7] to non-convex ones.

2.1. Integration of homogeneous functions

We consider the problem of evaluating integrals of the form $\int_{\mathcal{P}} f(\mathbf{x})\,\mathrm{d}\mathbf{x}$, where:

- $f : \mathcal{P} \to \mathbb{R}$ denotes a *(positively) homogeneous function* of degree $q \in \mathbb{R}$; that is, for all $\lambda > 0$ we have that $f(\lambda\mathbf{x}) = \lambda^q f(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{P}$. The function f satisfies Euler's homogeneous function theorem [22]: If f is a continuously differentiable positively homogeneous function of degree q, then we have
$$qf(\mathbf{x}) = \mathbf{x} \cdot \nabla f(\mathbf{x})$$
for all $\mathbf{x}$ in the domain of definition of f.
- $\mathcal{P} \subset \mathbb{R}^d$, $d = 2, 3$, denotes a closed polytope whose boundary $\partial\mathcal{P}$ is defined by $m(\mathcal{P})$ polytopic facets $\{\mathcal{F}_i\}_{i=1}^{m(\mathcal{P})}$ of dimension $(d-1)$. To each facet $\mathcal{F}_i$, we associate the hyperplane $\mathcal{H}_i$ containing $\mathcal{F}_i$ defined for any $\mathbf{x}_0 \in \mathbb{R}^d$ by
$$\mathcal{H}_i = \{\mathbf{x} \in \mathbb{R}^d : (\mathbf{x} - \mathbf{x}_0) \cdot \mathbf{n}_i = a_i\}$$
for some $a_i \in \mathbb{R}$ and some vector $\mathbf{n}_i \in \mathbb{R}^d$ of unit length. We note that $\mathbf{n}_i$ may be chosen to be the unit outward normal to $\mathcal{P}$ on $\mathcal{F}_i$ and that a_i denotes the (signed) distance between $\mathcal{H}_i$ and $\mathbf{x}_0$.

We also recall the generalised Stokes' theorem [26]: For a continuously differentiable vector field $\mathbf{X} : \mathcal{P} \to \mathbb{R}^d$ defined on a neighbourhood of $\mathcal{P}$, we have
$$\int_{\mathcal{P}} \nabla \cdot \mathbf{X}\,\mathrm{d}\mathbf{x} = \int_{\partial\mathcal{P}} \mathbf{X} \cdot \mathbf{n}\,\mathrm{d}\sigma,$$
where $\mathrm{d}\sigma$ denotes the Lebesgue measure on $\partial\mathcal{P}$ and $\mathbf{n}$ denotes the unit outward normal to $\mathcal{P}$ on $\partial\mathcal{P}$.

By setting $\mathbf{X} = (\mathbf{x} - \mathbf{x}_0)f(\mathbf{x})$ and invoking Euler's homogeneous function theorem, it can be shown that
$$\int_{\mathcal{P}} f(\mathbf{x})\,\mathrm{d}\mathbf{x} = \frac{1}{d+q}\left[\sum_{i=1}^{m(\mathcal{P})} a_i \int_{\mathcal{F}_i} f(\mathbf{x})\,\mathrm{d}\sigma + \int_{\mathcal{P}} \mathbf{x}_0 \cdot \nabla f(\mathbf{x})\ \mathrm{d}\mathbf{x}\right]. \tag{2.1}$$
Equation (2.1) relates integration of f over $\mathcal{P}$ in terms of integration of f over the facets of $\mathcal{P}$, as well as the integration of ∇f over $\mathcal{P}$. By selecting $\mathbf{x}_0 = \mathbf{0}$, (2.1) reduces to the more common expression relating the integrals of f over $\mathcal{P}$ and $\partial\mathcal{P}$ [2,7,8,18]:
$$\int_{\mathcal{P}} f(\mathbf{x})\,\mathrm{d}\mathbf{x} = \frac{1}{d+q}\sum_{i=1}^{m(\mathcal{P})} a_i \int_{\mathcal{F}_i} f(\mathbf{x})\,\mathrm{d}\sigma.$$

As shown in [2,7,17,18], for example, one may recursively apply the generalised Stokes' theorem to express the integral $\int_{\mathcal{F}_i} f(\mathbf{x})\,\mathrm{d}\sigma$ in terms of integrals over the $(d-2)$-dimensional boundary facets $\{\mathcal{F}_{ij}\}_{j=1}^{m(\mathcal{F}_i)}$ of $\mathcal{F}_i$:
$$\int_{\mathcal{F}_i} f(\mathbf{x})\,\mathrm{d}\sigma = \frac{1}{d-1+q}\left[\sum_{j=1}^{m(\mathcal{F}_i)} a_{ij} \int_{\mathcal{F}_{ij}} f(\mathbf{x})\ \mathrm{d}\nu + \int_{\mathcal{F}_i} \mathbf{x}_1 \cdot \nabla f(\mathbf{x})\ \mathrm{d}\sigma\right], \tag{2.2}$$
where $\mathrm{d}\nu$ denotes the Lebesgue measure on $\partial\mathcal{F}_i$, $\mathbf{x}_1 \in \mathcal{H}_i$ is arbitrary and a_{ij} denotes the Euclidean distance from $\mathbf{x}_1$ to $\mathcal{F}_{ij}$.

Equations (2.1) and (2.2) can be generalised to give the integral of f over any k-dimensional facet $\mathcal{F}$, $0 \leq k \leq d$, in terms of the integral of the same function over the boundary $\partial\mathcal{F} = \{\partial\mathcal{F}_i\}_{i=1}^{m(\mathcal{F})}$ and the integral of ∇f over $\mathcal{F}$:

$$\int_{\mathcal{F}} f(\mathbf{x})\,\mathrm{d}s = \frac{1}{\dim\mathcal{F} + q}\left[\sum_{i=1}^{m(\mathcal{F})} \operatorname{dist}(\partial\mathcal{F}_i, \mathbf{x}_{\mathcal{F}}) \int_{\partial\mathcal{F}_i} f(\mathbf{x})\,\mathrm{d}\xi + \int_{\mathcal{F}} \mathbf{x}_{\mathcal{F}} \cdot \nabla f(\mathbf{x})\,\mathrm{d}s\right], \tag{2.3}$$

where $\mathrm{d}s$ (respectively $\mathrm{d}\xi$) denotes the k-dimensional (respectively $(k-1)$-dimensional) Lebesgue measure on $\mathcal{F}$ (respectively $\partial\mathcal{F}$), $\mathbf{x}_{\mathcal{F}}$ is an arbitrary point contained in $\mathcal{F}$ (or the k-dimensional hyperplane containing $\mathcal{F}$), and $\operatorname{dist}(\partial\mathcal{F}_i, \mathbf{x}_{\mathcal{F}})$ denotes the Euclidean distance from $\mathbf{x}_{\mathcal{F}}$ to (the k-dimensional hyperplane containing) $\partial\mathcal{F}_i$. Finally, in the case where $\mathcal{F} = \mathbf{x}_{\mathcal{F}} \in \mathbb{R}^d$ (that is, $\dim\mathcal{F} = 0$), the right-hand-side of (2.3) can be replaced with the point evaluation $f(\mathbf{x}_{\mathcal{F}})$.

In the case where $f(\mathbf{x}) = \mathbf{x}^{\boldsymbol{\alpha}} = \prod_{k=1}^{d} x_k^{\alpha_k}$ is a monomial function in d variables, (2.3) gives the following recursive formula for the integrals

$$\mathcal{I}(\mathcal{F}, \boldsymbol{\alpha}) := \int_{\mathcal{F}} \mathbf{x}^{\boldsymbol{\alpha}}\,\mathrm{d}s = \frac{1}{\dim\mathcal{F} + |\boldsymbol{\alpha}|}\left[\sum_{i=1}^{m(\mathcal{F})} \operatorname{dist}(\partial\mathcal{F}_i, \mathbf{x}_{\mathcal{F}})\mathcal{I}(\partial\mathcal{F}_i, \boldsymbol{\alpha}) + \sum_{j=1}^{d} \alpha_j (\mathbf{x}_{\mathcal{F}})_j \mathcal{I}(\mathcal{F}, \boldsymbol{\alpha} - \mathbf{e}_j)\right], \tag{2.4}$$

where $\mathbf{e}_i$, $i = 1, 2, \ldots, d$, denote the standard unit basis vectors in $\mathbb{R}^d$.

Equation (2.4) can be used to generate sets of integrals of monomial functions over $\mathcal{P}$. To this end, we consider the problem of evaluating the following set of integrals:

$$\mathcal{I}(\mathcal{F}, \mathcal{J}) = \left\{\mathcal{I}(\mathcal{F}, \boldsymbol{\alpha}) = \int_{\mathcal{F}} \mathbf{x}^{\boldsymbol{\alpha}}\,\mathrm{d}\mathbf{x} : \boldsymbol{\alpha} \in \mathcal{J}\right\}.$$

Here, $\mathcal{J} \subset \mathbb{N}_0^d$ denotes a set of multi-indices satisfying the following property: For each $\boldsymbol{\alpha} \in \mathcal{J}$ and $1 \leq i \leq d$, we either have $\alpha_i = 0$ or $\boldsymbol{\alpha} - \mathbf{e}_i \in \mathcal{J}$. This motivates the definition of Algorithm 1, cf. [2].

Algorithm 1 Evaluation of the set $\mathcal{I}(\mathcal{F}, \mathcal{J})$ for a given k-dimensional facet $\mathcal{F}$, $0 \leq k \leq d$.

```
1: procedure ComputeIntegrals($\mathcal{F}$,$\mathcal{J}$)
2:     $\mathcal{I}(\mathcal{F}, \boldsymbol{\alpha}) \leftarrow 0$ for all $\boldsymbol{\alpha} \in \mathcal{J}$
3:     Select $\mathbf{x}_{\mathcal{F}}$ as any point in (the ($\dim\mathcal{F}$)-dimensional hyperplane containing) $\mathcal{F}$
4:     Get boundary facets $\{\partial\mathcal{F}_i\}_{i=1}^{m(\mathcal{F})}$
5:     for $i = 1, \ldots, m(\mathcal{F})$ do
6:         if $\operatorname{dist}(\partial\mathcal{F}_i, \mathbf{x}_{\mathcal{F}}) \neq 0$ and $\mathcal{I}(\partial\mathcal{F}_i, \mathcal{J})$ not already computed then
7:             $\mathcal{I}(\partial\mathcal{F}_i, \mathcal{J}) \leftarrow$ ComputeIntegrals($\partial\mathcal{F}_i, \mathcal{J}$)
8:         end if
9:     end for
10:    for $\boldsymbol{\alpha} \in \mathcal{J}$ do
11:        $$\mathcal{I}(\mathcal{F}, \boldsymbol{\alpha}) \leftarrow \frac{1}{\dim\mathcal{F} + |\boldsymbol{\alpha}|}\left[\sum_{i=1}^{m(\mathcal{F})} \operatorname{dist}(\partial\mathcal{F}_i, \mathbf{x}_{\mathcal{F}})\mathcal{I}(\partial\mathcal{F}_i, \boldsymbol{\alpha}) + \sum_{j=1}^{d} \alpha_j (\mathbf{x}_{\mathcal{F}})_j \mathcal{I}(\mathcal{F}, \boldsymbol{\alpha} - \mathbf{e}_j)\right].$$
12:    end for
13:    return $\mathcal{I}(\mathcal{F}, \mathcal{J})$
14: end procedure
```

Remark 1 (Termination of Algorithm 1)**.** *The recursion in Algorithm 1 terminates when* ComputeIntegrals($\mathcal{F}, \mathcal{J}$) *is called with* $\dim \mathcal{F} = 0$*; that is,* $\mathcal{F} = \mathbf{x}_{\mathcal{F}}$ *is a single point. Since* $\partial\mathcal{F} = \mathbf{x}_{\mathcal{F}}$*, the recursive function call in line 7 will not be executed.*

2.2. Analysis of quadrature-free monomial integration algorithm

The computational complexity of Algorithm 1 can be understood in terms of the size of the requested monomial set $\mathcal{J}$, as well as the complexity of the domain of integration $\mathcal{P}$. With this in mind, the main result of this article is stated below.

Theorem 1 (Time complexity of Algorithm 1)**.** *The time complexity of Algorithm 1 including the evaluation of the set*

$$\{\mathrm{dist}(\partial\mathcal{F}_i, \mathbf{x}_{\mathcal{F}}) : 1 \le i \le m(\mathcal{F}), 0 \le \dim \mathcal{F} \le d\}$$

is $O(\chi_1(\mathcal{P})|\mathcal{J}| + \chi_3(\mathcal{P}))$*, where*

$$\begin{aligned} \chi_1(\mathcal{P}) &= \sum_{k=0}^{d} \sum_{\substack{\mathcal{F}\subseteq\mathcal{P} \\ \dim\mathcal{F}=k}} (m(\mathcal{F}) + d), \\ \chi_3(\mathcal{P}) &= d\sum_{k=0}^{d} \sum_{\substack{\mathcal{F}\subseteq\mathcal{P} \\ \dim\mathcal{F}=k}} k^2 m(\mathcal{F}), \end{aligned}$$

and $|\mathcal{J}|$ *denotes the number of requested monomial integrals.*

The proof of Theorem 1 will be pursued in the following two sections: Firstly, in Section 2.2.1 we consider both the time and space complexity of Algorithm 1 in the case when the required set of distance functions is pre-computed; the cost of the evaluation of this latter set is then determined in Section 2.2.2.

2.2.1. Complexity of Algorithm 1 with pre-computed distances

In this section we study the computational cost of computing $\mathcal{I}(\mathcal{P}, \mathcal{J})$ using Algorithm 1 in the case when the desired set of distance functions is pre-computed; here, $\mathcal{I}(\mathcal{P}, \mathcal{J})$ is defined in an analogous manner to $\mathcal{I}(\mathcal{F}, \mathcal{J})$, with $\mathcal{F}$ replaced by $\mathcal{P}$ in (2.4). The main result of this section is given in Lemma 1 below; furthermore we also consider potential simplifications to Algorithm 1 which can be utilised to further reduce the computational expense.

Lemma 1 (Time and space complexity of Algorithm 1)**.** *Assuming that the set*

$$\{\mathrm{dist}(\partial\mathcal{F}_i, \mathbf{x}_{\mathcal{F}}) : 1 \le i \le m(\mathcal{F}), 0 \le \dim \mathcal{F} \le d\}$$

is pre-computed, the time complexity of Algorithm 1, measured as the total number of floating-point operations required to assemble $\mathcal{I}(\mathcal{P}, \mathcal{J})$*, is* $O(\chi_1(\mathcal{P})|\mathcal{J}|)$*, where* $|\mathcal{J}|$ *denotes the number of requested monomial integrals and*

$$\chi_1(\mathcal{P}) = \sum_{k=0}^{d} \sum_{\substack{\mathcal{F}\subseteq\mathcal{P} \\ \dim\mathcal{F}=k}} (m(\mathcal{F}) + d).$$

The space complexity of Algorithm 1, measured as the total number of floating-point numbers required to store $\mathcal{I}(\mathcal{P},\mathcal{J})$, is $O(\chi_2(\mathcal{P})|\mathcal{J}|)$, where

$$\chi_2(\mathcal{P}) = \sum_{k=0}^{d} card\{\mathcal{F} \subseteq \mathcal{P} : \dim \mathcal{F} = k\}.$$

Proof. We first analyse the number of floating-point operations required to compute the right-hand side of (2.4) for a single facet $\mathcal{F}$:

- The sum
$$S_1 = \sum_{i=1}^{m(\mathcal{F})} \operatorname{dist}(\partial\mathcal{F}_i, \mathbf{x}_{\mathcal{F}})\mathcal{I}(\partial\mathcal{F}_i, \boldsymbol{\alpha})$$
requires $m(\mathcal{F})$ products and $m(\mathcal{F}) - 1$ additions;
- The sum
$$S_2 = \sum_{j=1}^{d} \alpha_j(\mathbf{x}_{\mathcal{F}})_j\mathcal{I}(\mathcal{F}, \boldsymbol{\alpha} - \mathbf{e}_j)$$
requires $2d$ products and $d - 1$ additions;
- The sum
$$S_3 = \dim \mathcal{F} + |\boldsymbol{\alpha}| = \dim \mathcal{F} + \sum_{j=1}^{d} \alpha_j$$
requires d additions;
- The final result $\mathcal{I}(\mathcal{F}, \boldsymbol{\alpha}) = \frac{S_1+S_2}{S_3}$ requires one addition and one division.

We deduce that the right-hand-side of (2.4) may be computed in $2m(\mathcal{F}) + 4d$ floating-point operations. Since this operation is called for each $\boldsymbol{\alpha} \in \mathcal{J}$, lines 10–12 of ComputeIntegrals requires $c_{\mathcal{F}} = (2m(\mathcal{F}) + 4d)|\mathcal{J}|$ floating-point operations.

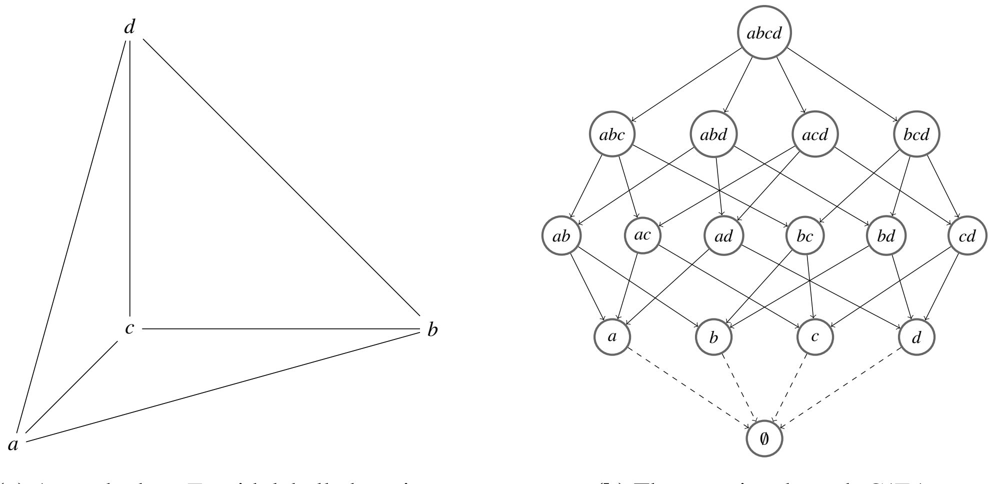
It is not difficult to see that ComputeIntegrals is executed exactly once for each k-dimensional facet $\mathcal{F} \subseteq \mathcal{P}$ with $0 \leq k \leq d$. Thus, summing $c_{\mathcal{F}}$ over each $\mathcal{F}$, the time complexity given in the statement of the theorem is proven. The space complexity can be proven based on noting that $\mathcal{I}(\mathcal{F},\mathcal{J})$ is a set with $|\mathcal{J}|$ elements which must be stored for each facet $\mathcal{F} \subseteq \mathcal{P}$. □

Remark 2 (Simplifications for $\dim(\mathcal{F}) = d$ and $\dim(\mathcal{F}) = 0$)**.** *When* $\dim(\mathcal{F}) = d$*, any selection of* $\mathbf{x}_{\mathcal{F}} \in \mathbb{R}^d$ *can be made. By choosing* $\mathbf{x}_{\mathcal{F}} = \mathbf{0}$*, the second sum in (2.4) is eliminated and lines 10–12 in Algorithm 1 can be performed in* $(2m(\mathcal{F}) + d)|\mathcal{J}|$ *floating-point operations.*

When $\dim(\mathcal{F}) = 0$ *(i.e.,* $\mathcal{F} = \mathbf{x}_{\mathcal{F}} \in \mathbb{R}^d$*),* $\mathcal{I}(\mathcal{F}, \boldsymbol{\alpha})$ *can be performed in a couple of ways:*

- *Direct computation of* $\mathcal{I}(\mathcal{F}, \boldsymbol{\alpha}) = \prod_{k=1}^{d}(\mathbf{x}_{\mathcal{F}})_k^{\alpha_k}$*, in this case, line 11 of Algorithm 1 can be performed in* $O(d + \sum_{k=1}^{d} \log(1 + \alpha_i))$ *floating-point operations via binary exponentiation [16].*
- *Recursive computation of* $\mathcal{I}(\mathcal{F}, \boldsymbol{\alpha}) = \alpha_k(\mathbf{x}_{\mathcal{F}})_k\mathcal{I}(\mathcal{F}, \boldsymbol{\alpha} - \mathbf{e}_k)$ *for some* $1 \leq k \leq d$*, in this case, line 11 of Algorithm 1 can be performed in* $O(1)$ *floating-point operations.*

The scalings of the time and space complexities reported in Lemma 1 as functions of the geometric complexity of $\mathcal{P}$ can be understood by visualising the recursive nature of Algorithm 1 as a directed acyclic graph $G = G(\mathcal{P}) = (V, E)$. The vertex set $V = V(\mathcal{P})$ is defined as the set of k-dimensional facets $\mathcal{F} \subseteq \mathcal{P}$ for $0 \leq k \leq d$. The edge set $E = E(\mathcal{P})$ is defined as follows: For any facets $\mathcal{F}_1, \mathcal{F}_2 \in V$, the directed edge $(\mathcal{F}_1, \mathcal{F}_2) \in E$ if and only if $\dim \mathcal{F}_2 = \dim \mathcal{F}_1 - 1$ and $\mathcal{F}_2$ lies on the boundary of $\mathcal{F}_1$. Figure 1 gives an example of the construction of $G(\mathcal{P})$.



(a) A tetrahedron T_3 with labelled vertices. **(b)** The associated graph $G(T_3)$.

Figure 1. Example of a tetrahedron $\mathcal{P} = T_3$ (left) and the associated recursive call graph $G(\mathcal{P})$ (right). Each vertex of $G(\mathcal{P})$ represents a facet of the tetrahedron (e.g., a vertex, an edge or a face). Edges between vertices in $G(\mathcal{P})$ denote the relationship between facets on the boundaries of other facets (e.g., the edge *ab* lies on the boundary of the face *abc*).

Remark 3 (Facet lattice of $\mathcal{P}$)**.** *It is convenient to add an extra vertex $\emptyset$ to $V(\mathcal{P})$, which we define as having dimension -1, and extra directed edges to $E(\mathcal{P})$ from each vertex of $\mathcal{P}$ to $\emptyset$. The resulting graph $G(\mathcal{P})$ then resembles a Hasse diagram representing the facet lattice formed from the facets of $\mathcal{P}$ ordered by inclusion [4, 14].*

Remark 4 (Algorithm 1 as a depth-first search)**.** *The recursion of Algorithm 1 can be understood as a depth-first search of the graph $G(\mathcal{P})$ starting at $\mathcal{P}$ where $\mathcal{I}(\mathcal{F}, \mathcal{J})$ is assembled at each unvisited face $\mathcal{F}$. By contrast, the implementation of Algorithm 1 given in [8] can be understood as a breadth-first search of the transpose graph $G'(\mathcal{P})$ starting at $\emptyset$, where $G'(\mathcal{P})$ differs from $G(\mathcal{P})$ only by reversal of the directed edges.*

The time complexity of Algorithm 1 can be understood in terms of the sizes of the vertex and edge sets $V(\mathcal{P})$ and $E(\mathcal{P})$. In particular, we have that

$$\sum_{k=0}^{d} \sum_{\substack{\mathcal{F} \subset \mathcal{P} \\ \dim \mathcal{F} = k}} (2m(\mathcal{F}) + 4d) = 2 \sum_{k=0}^{d} \sum_{\substack{\mathcal{F} \subset \mathcal{P} \\ \dim \mathcal{F} = k}} m(\mathcal{F}) + 4d \sum_{k=0}^{d} \operatorname{card}\{\mathcal{F} \subseteq \mathcal{P} : \dim \mathcal{F} = k\} = 2|E(\mathcal{P})| + 4d|V(\mathcal{P})|$$

and the time and space complexities reported in Lemma 1 can be alternatively be written as $\mathcal{O}((|E(\mathcal{P})| + d|V(\mathcal{P})|)|\mathcal{J}|)$ and $\mathcal{O}(|V(\mathcal{P})||\mathcal{J}|)$, respectively. Table 1 gives the time and space complexities of Algorithm 1 for a number of different classes of polytopes, as well as the sizes of the vertex and edge sets $V(\mathcal{P})$ and $E(\mathcal{P})$ in the graph $G(\mathcal{P})$, respectively.

In practical finite element applications, $\mathcal{P}$ may denote a d-dimensional mesh element, $d = 2, 3$, in a polytopic mesh $\mathcal{T}$ of some domain of interest Ω. Historically, simplicial or tensor-product elements have been used to partition the domain, though general polytopic elements have been proposed more recently, cf. [1, 3, 5, 6, 9], for example. The following theorem characterises the time and space complexities of Algorithm 1 in these special cases.

Table 1. Time and space complexities of Algorithm 1 for different polytopes $\mathcal{P}$ as well as the set sizes $|V(\mathcal{P})|$ and $|E(\mathcal{P})|$ under the assumption that an extra vertex $\emptyset$ is added to $V(\mathcal{P})$.

Family	$\lvert V(\mathcal{P})\rvert$	$\lvert E(\mathcal{P})\rvert$	Time complexity	Space complexity
d-dimensional simplex	2^{d+1}	$2^d(d+1)$	$O(2^d d\lvert\mathcal{J}\rvert)$	$O(2^d\lvert\mathcal{J}\rvert)$
d-dimensional hypercube	3^d+1	$2\cdot 3^{d-1}d+2^d$	$O(3^d d\lvert\mathcal{J}\rvert)$	$O(3^d\lvert\mathcal{J}\rvert)$
n-sided polygon	$2(n+1)$	$4n$	$O(n\lvert\mathcal{J}\rvert)$	$O(n\lvert\mathcal{J}\rvert)$
n-gonal prism	$2(3n+2)$	$15n+2$	$O(n\lvert\mathcal{J}\rvert)$	$O(n\lvert\mathcal{J}\rvert)$
n-based pyramid	$4(n+1)$	$2(5n+1)$	$O(n\lvert\mathcal{J}\rvert)$	$O(n\lvert\mathcal{J}\rvert)$

Lemma 2 (Complexity of Algorithm 1 for $d = 2, 3$)**.** *Let $\mathcal{P} \subset \mathbb{R}^d$, $d = 2, 3$, denote a convex polygon or polyhedron. The time and space complexities of Algorithm 1, measured in the sense described in Lemma 1, are $O(e|\mathcal{J}|)$, where e denotes the number of (1-dimensional) edges of $\mathcal{P}$.*

Proof. It suffices to show that $|V(\mathcal{P})| = O(e)$ and $|E(\mathcal{P})| = O(e)$.

For the case $d = 2$, a polygon $\mathcal{P}$ with e edges also has e vertices. Therefore, we have that $|V(\mathcal{P})| = 2(e+1)$ and $|E(\mathcal{P})| = 4e$, as required.

For the case $d = 3$, let v (respectively f) denote the number of vertices (respectively number of faces) of $\mathcal{P}$. Since $\mathcal{P}$ is convex, the Euler characteristic of the surface of $\mathcal{P}$ is equal to 2 [12], and hence

$$v - e + f = 2.$$

The size of the vertex set $V(\mathcal{P})$ is given by

$$|V(\mathcal{P})| = v + e + f + 2 = 2(e + 2).$$

To compute the size of the edge set $E(\mathcal{P})$, we note that each edge (or 1-dimensional facet) $\mathcal{F}$ in $G(\mathcal{P})$ has in-degree and out-degree 2, since each edge lies on the boundary of two faces and has two vertices on its own boundary. We therefore have that

$$|E(\mathcal{P})| = v + 2e + 2e + f = 5e + 2.$$

$\square$

Remark 5 (Extension to non-convex polyhedra)**.** *The argument presented in the proof of Lemma 2 remains valid when both $\mathcal{P}$ and its $(d-1)$-dimensional facets $\{\mathcal{F}_i\}_{i=1}^{m(\mathcal{P})}$ are simply-connected. That is, Lemma 2 holds if neither $\mathcal{P}$ nor any of its facets have any holes.*

One may reduce the time and space complexities of Algorithm 1 through judicious selection of the reference points $\mathbf{x}_{\mathcal{F}}$. When $\mathbf{x}_{\mathcal{F}}$ is chosen as a vertex of $\mathcal{F}$, one may avoid a number of recursive calls to ComputeIntegrals for some boundary facets of $\mathcal{F}$. We shall refer to the resulting implementation as *pruned* - this is illustrated in Figure 2. While a complete time and space complexity analysis of Algorithm 1 with pruning is not presented here, pruning can lead to significant computational savings on simple geometries. For instance, the time complexity of Algorithm 1 in the case where $\mathcal{P}$ is a d-dimensional simplex is $O(2^d d|\mathcal{J}|)$; with pruning, this can be reduced to $O(d^2|\mathcal{J}|)$. On more complicated domains, the effects of pruning are likely to be less significant. To our knowledge, finding an optimal pruning of $G(\mathcal{P})$ for a general polytope, that is, a shortest sequence of visited facets $(\mathcal{F}_n)_{n\geq 0}$ and corresponding reference points $(\mathbf{x}_{\mathcal{F}_n})_{n\geq 0}$ for which Algorithm 1 can compute $\mathcal{I}(\mathcal{P},\mathcal{J})$ - is an open problem.

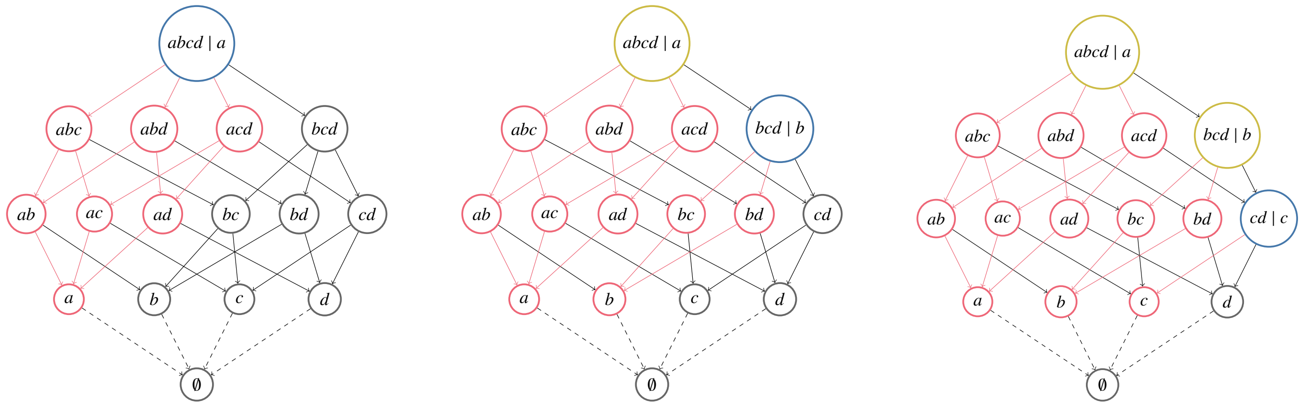


Figure 2. Effect of pruning on the recursive call graph for Algorithm 1 in the case $\mathcal{P} = T_3$ as in Figure 1. Left-to-right: first three recursive executions of ComputeIntegrals. Blue node: facet $\mathcal{F}$ associated with current execution of ComputeIntegrals($\mathcal{F},\mathcal{J}$) and choice of reference point $\mathbf{x}_{\mathcal{F}}$. Yellow nodes: facets $\mathcal{F}$ associated with previous executions of ComputeIntegrals($\mathcal{F},\mathcal{J}$) and choice of reference point $\mathbf{x}_{\mathcal{F}}$. Red nodes: facets $\mathcal{F}$ eliminated from recursion as a result of pruning; i.e., unvisited facets with $\operatorname{dist}(\mathcal{F},\mathbf{x}_{\mathcal{F}'}) = 0$ for some previously-selected reference point $\mathbf{x}_{\mathcal{F}'}$.

2.2.2. Time complexity of distance pre-computation

It is convenient to omit the computation of distances of the form $\operatorname{dist}(\partial\mathcal{F}_i,\mathbf{x}_{\mathcal{F}})$ in the proof of Lemma 1 since such quantities do not need to be re-computed for each $\mathcal{I}(\mathcal{F},\boldsymbol{\alpha})$. In cases where $|\mathcal{J}|$ is very small or $\mathcal{P}$ has many facets, the evaluation of these distances can become the most computationally-expensive part of Algorithm 1. For instance, it is shown in [4] that the time complexity of the computation of the volume of a d-dimensional hypercube via the quadrature-free integration method is $O(d^4 3^d)$, which is a factor of $O(d^3)$ larger than that predicted by Lemma 1. To remedy this, the following lemma measures the complexity of operations omitted in the proof of Lemma 1.

Lemma 3 (Time complexity of distance pre-computation)**.** *The time complexity of evaluating the set*

$$\{\operatorname{dist}(\partial\mathcal{F}_i,\mathbf{x}_{\mathcal{F}}) : 1 \leq i \leq m(\mathcal{F}), 0 \leq \dim\mathcal{F} \leq d\}$$

is $O(\chi_3(\mathcal{P}))$, where

$$\chi_3(\mathcal{P}) = d\sum_{k=0}^{d}\sum_{\substack{\mathcal{F}\subseteq\mathcal{P}\\ \dim\mathcal{F}=k}} k^2 m(\mathcal{F}).$$

Alternatively, the time complexity may be expressed as $O(d^3|E(\mathcal{P})|)$.

Proof. Let $\mathcal{F}$ denote a k-dimensional facet with selected reference point $\mathbf{x}_{\mathcal{F}}$. Let $\partial\mathcal{F}$ denote any of its $(k-1)$-dimensional boundary facets. Suppose that $\partial\mathcal{F}$ has n vertices $\{\mathbf{x}_i\}_{i=0}^{n-1}$ and note that $k \leq n$. The $(k-1)$-dimensional hyperplane $\mathcal{H}$ containing $\partial\mathcal{F}$ can be uniquely defined by the points $\{\mathbf{x}_i\}_{i=0}^{k-1}$: For any $\mathbf{x} \in \mathcal{H}$, there exists $\mathbf{t} = (t_i)_{i=1}^{k-1} \in \mathbb{R}^{k-1}$ such that

$$\mathbf{x} = \mathbf{x}(\mathbf{t}) = \mathbf{x}_0 + \sum_{i=1}^{k-1}(\mathbf{x}_i - \mathbf{x}_0)t_i.$$

The distance $\operatorname{dist}(\partial\mathcal{F}, \mathbf{x}_{\mathcal{F}})$ (or, more precisely, the distance between $\mathcal{H}$ and $\mathbf{x}_{\mathcal{F}}$) is the minimum value of $\|\mathbf{x}(\mathbf{t}) - \mathbf{x}_{\mathcal{F}}\|_2$ over $\mathbf{t} \in \mathbb{R}^{k-1}$; this occurs when $\mathbf{t}$ solves

$$\mathbf{A}\mathbf{t} = \mathbf{f},$$

where the entries of $\mathbf{A} \in \mathbb{R}^{(k-1)\times(k-1)}$ and $\mathbf{f} \in \mathbb{R}^{k-1}$ are given by $(\mathbf{A})_{ij} = (\mathbf{x}_i - \mathbf{x}_0)\cdot(\mathbf{x}_j - \mathbf{x}_0)$ and $(\mathbf{f})_i = (\mathbf{x}_0 - \mathbf{x}_{\mathcal{F}})\cdot(\mathbf{x}_i - \mathbf{x}_0)$, respectively. The number of floating-point operations required to assemble and solve the linear system above is $O(k^2 d)$ and $O(k^3)$, respectively; thus, the time complexity of computing $\operatorname{dist}(\partial\mathcal{F}, \mathbf{x}_{\mathcal{F}})$ for a single $(k-1)$-dimensional facet $\partial\mathcal{F}$ is $O(k^2 d)$.

For each k-dimensional facet $\mathcal{F}$, $0 \leq k \leq d$, $\operatorname{dist}(\partial\mathcal{F}_i, \mathbf{x}_{\mathcal{F}})$ is computed for each $1 \leq i \leq m(\mathcal{F})$. Summing the time complexity of a single computation of $\operatorname{dist}(\partial\mathcal{F}, \mathbf{x}_{\mathcal{F}})$ over these limits yields the first result stated in the proof; the second result is obtained by bounding the complexity of a single computation of $\operatorname{dist}(\partial\mathcal{F}, \mathbf{x}_{\mathcal{F}})$ from above by $O(d^3)$. □

The proof of Theorem 1 now follows immediately by combining Lemma 1, together with Lemma 3. Furthermore, employing the notation introduced in Section 2.2.1, the statement of Theorem 1 may be rewritten in the following compact form.

Corollary 1 (Time complexity of Algorithm 1)**.** *The time complexity of Algorithm 1 including the evaluation of the set*

$$\{\operatorname{dist}(\partial\mathcal{F}_i, \mathbf{x}_{\mathcal{F}}) : 1 \leq i \leq m(\mathcal{F}), 0 \leq \dim\mathcal{F} \leq d\}$$

is $O\left(d^3|E(\mathcal{P})| + (d|V(\mathcal{P})| + |E(\mathcal{P})|)\,|\mathcal{J}|\right)$.

3. DGFEM discretisation of the transport equation

In this section we consider the application of the numerical integration algorithm outlined in the previous section for the computation of the volume integrals arising within the DGFEM discretisation of the linear transport equation. To this end, given an open bounded polyhedral domain $\Omega \subset \mathbb{R}^d$, $d = 2, 3$, we consider the following advection-reaction equation: Find $u : \Omega \to \mathbb{R}$ such that

$$\begin{aligned} \nabla\cdot(\mathbf{b}u) + cu &= f \quad \text{in } \Omega, \\ u &= g \quad \text{on } \Gamma_{in}, \end{aligned} \tag{3.1}$$

where $c, f : \Omega \to \mathbb{R}$, $g : \Gamma_{in} \to \mathbb{R}$ and $\mathbf{b} : \Omega \to \mathbb{R}^d$ are given data terms, and $\Gamma_{in} = \{\mathbf{x} \in \partial\Omega : \mathbf{b}\cdot\mathbf{n}(\mathbf{x}) < 0\}$ denotes the inflow boundary of Ω, where $\mathbf{n}(\mathbf{x})$ denotes the outward unit normal to Ω at $\mathbf{x} \in \partial\Omega$. In the following, we assume that $\mathbf{b}$ is a constant velocity vector, while c is a given as a piecewise-constant function with respect to the elements in the underlying finite element mesh $\mathcal{T}$ defined below. For the type of applications we have in mind, namely the numerical approximation of the linear Boltzmann transport problem, cf. [13], these assumptions are not restrictive. That said, the proposed quadrature-free implementation can easily be extended to include the case when $\mathbf{b}$, c, f and g are polynomial functions (or piecewise-polynomial with respect to the elements of $\mathcal{T}$).

3.1. Discretisation

We discretise the linear transport equation (3.1) using a DGFEM approach. To this end, let $\mathcal{T}$ denote a subdivision of the spatial domain Ω into open non-overlapping polytopic elements κ such that $\bar{\Omega} = \bigcup_{\kappa\in\mathcal{T}} \bar{\kappa}$. We denote by $\mathcal{E}$ the set of faces in $\mathcal{T}$, which are defined as the $(d-1)$-dimensional planar facets of elements $\kappa \in \mathcal{T}$.

To each $\kappa \in \mathcal{T}$ we respectively denote by $h_\kappa > 0$ and $p_\kappa \geq 0$ the diameter of κ and the polynomial degree on κ. The spatial finite element space is defined by

$$\mathbb{V} = \{v \in L_2(\Omega) : v|_\kappa \in \mathbb{P}^{p_\kappa}(\kappa) \text{ for all } \kappa \in \mathcal{T}\},$$

where $\mathbb{P}^{p_\kappa}(\kappa)$ denotes the space of all d-variate polynomials with maximum total degree at most p_κ on κ.

Given $\kappa \in \mathcal{T}$, we define the inflow and outflow parts of $\partial\kappa$ by

$$\begin{aligned}\partial_-\kappa &= \{\mathbf{x} \in \partial\kappa : \mathbf{b}\cdot\mathbf{n}(\mathbf{x}) < 0\},\\ \partial_+\kappa &= \{\mathbf{x} \in \partial\kappa : \mathbf{b}\cdot\mathbf{n}(\mathbf{x}) \geq 0\},\end{aligned}$$

respectively, where $\mathbf{n}(\mathbf{x})$ denotes the outward unit normal to κ at $\mathbf{x} \in \partial\kappa$. For a sufficiently-smooth function v, we denote by v_κ^+ (respectively, v_κ^-) the interior (respectively, exterior) trace of v on $\partial\kappa$ (respectively, $\partial\kappa \setminus \partial\Omega$). Since it will always be clear which element $\kappa \in \mathcal{T}$ the quantities $v_\kappa^\pm$ correspond to, the subscript κ will be suppressed for the remainder of this article.

The DGFEM discretisation of (3.1) with upwind numerical flux reads as follows: Find $u_h \in \mathbb{V}$ such that

$$a(u_h, v_h) = \ell(v_h) \tag{3.2}$$

for all $v_h \in \mathbb{V}$, where $a : \mathbb{V}\times\mathbb{V} \to \mathbb{R}$ and $\ell : \mathbb{V} \to \mathbb{R}$ are defined, respectively, for all $w_h, v_h \in \mathbb{V}$ by

$$\begin{aligned}a(w_h, v_h) &= \sum_{\kappa\in\mathcal{T}} \left(\int_\kappa (-w_h \mathbf{b}\cdot\nabla v_h + c w_h v_h)\,\mathrm{d}\mathbf{x} + \int_{\partial_+\kappa} |\mathbf{b}\cdot\mathbf{n}|\, w_h^+ v_h^+ \,\mathrm{d}s - \int_{\partial_-\kappa\setminus\partial\Omega} |\mathbf{b}\cdot\mathbf{n}|\, w_h^- v_h^+ \,\mathrm{d}s \right),\\ \ell(v_h) &= \sum_{\kappa\in\mathcal{T}} \left(\int_\kappa f v_h \,\mathrm{d}\mathbf{x} + \int_{\partial_-\kappa\cap\partial\Omega} |\mathbf{b}\cdot\mathbf{n}|\, g v_h^+ \,\mathrm{d}s \right).\end{aligned}$$

3.2. Basis functions on polytopic elements

For the remainder of this article we will assume that each element $\kappa \in \mathcal{T}$ is equipped with a basis comprising of the polynomial space $\mathbb{P}^p(\kappa)$ for some fixed $p \geq 0$. Following [2, 6], we construct a Cartesian bounding box B_κ for each element $\kappa \in \mathcal{T}$ such that $\bar{\kappa} \subseteq \bar{B}_\kappa$. Furthermore, we define a reference bounding box $\hat{B} = (-1,1)^d$ and a family of affine mappings $F_\kappa : \hat{B} \to B_\kappa$ such that $F_\kappa(\hat{\mathbf{x}}) = \mathbf{J}_\kappa\hat{\mathbf{x}} + \mathbf{t}_\kappa$, where $\mathbf{J}_\kappa \in \mathbb{R}^{d\times d}$ is the (diagonal) Jacobi matrix of F_κ and $\mathbf{t}_\kappa \in \mathbb{R}^d$ is the translation between the origin $\mathbf{0} \in \hat{B}$ and the barycentre of B_κ.

We define a basis of $\mathbb{P}^p(\hat{B})$ as follows: We denote by $\{\mathcal{L}_n(t)\}_{n=0}^{\infty}$ the family of orthogonal (or orthonormal) univariate Legendre polynomials on $L_2(-1,1)$. For each multi-index $\boldsymbol{\alpha}$ of length d with $0 \le |\boldsymbol{\alpha}| \le p$ we define the basis function $\hat{\phi}_{\boldsymbol{\alpha}} : \hat{B} \to \mathbb{R}$ by

$$\hat{\phi}_{\boldsymbol{\alpha}}(\hat{\mathbf{x}}) = \prod_{k=1}^{d} \mathcal{L}_{\alpha_k}(\hat{x}_k). \tag{3.3}$$

It is straightforward to see that $\{\hat{\phi}_{\boldsymbol{\alpha}}(\hat{\mathbf{x}})\}_{0\le|\boldsymbol{\alpha}|\le p}$ forms a basis for $\mathbb{P}^p(\hat{B})$. The basis functions $\{\phi_{\boldsymbol{\alpha},\kappa}(\mathbf{x})\}_{0\le|\boldsymbol{\alpha}|\le p}$ for $\mathbb{P}^p(\kappa)$ are constructed upon application of the element mapping; more precisely, $\phi_{\boldsymbol{\alpha},\kappa}(\mathbf{x}) = \hat{\phi}_{\boldsymbol{\alpha}}(F_\kappa^{-1}(\mathbf{x}))$. The set

$$\{\phi_{\boldsymbol{\alpha},\kappa}(\mathbf{x}) : \kappa \in \mathcal{T}, 0 \le |\boldsymbol{\alpha}| \le p\}$$

forms a basis on each element κ, $\kappa \in \mathcal{T}$, for the finite element space $\mathbb{V}$. Henceforth, we will identify a bijection between the set of multi-indices $\{\boldsymbol{\alpha}\}_{0\le|\boldsymbol{\alpha}|\le p}$ and the set $\{1, \dots, \dim \mathbb{P}^p(\kappa)\}$ such that the i^{th} basis function on κ is denoted by $\phi_{\boldsymbol{\alpha}^{(i)},\kappa}(\mathbf{x})$.

We conclude this section by expanding the products of $\mathcal{L}_n(t)$ and their derivatives as sums of monomials:

$$\mathcal{L}_m(t)\mathcal{L}_n(t) = \sum_{k=0}^{m+n} C_{m,n,k} t^k, \tag{3.4}$$

$$\mathcal{L}'_m(t)\mathcal{L}_n(t) = \sum_{k=0}^{m+n-1} C'_{m,n,k} t^k. \tag{3.5}$$

The sets of coefficients $\{C_{m,n,k} : 0 \le m,n \le p, 0 \le k \le m+n\}$ and $\{C'_{m,n,k} : 0 \le m,n \le p, 0 \le k \le m+n-1\}$ may be pre-computed before assembly.

4. Analysis of volume matrix assembly

In this section we focus on the efficient assembly of the DGFEM matrix arising on the left-hand side of (3.2). Typically, when quadrature is employed, the evaluation of the volume integrals appearing in the left-hand side of (3.2) are far more expensive to compute than the corresponding face integrals since in the former case significantly more quadrature points must be employed. With that in mind, we focus on the acceleration of the assembly of the local element matrix contributions $\mathbf{A}_\kappa \in \mathbb{R}^{\dim \mathbb{P}^p(\kappa) \times \dim \mathbb{P}^p(\kappa)}$, where

$$(\mathbf{A}_\kappa)_{ij} = \int_\kappa \left(-\phi_{\boldsymbol{\alpha}^{(j)},\kappa}(\mathbf{x})\mathbf{b} \cdot \nabla\phi_{\boldsymbol{\alpha}^{(i)},\kappa}(\mathbf{x}) + c\phi_{\boldsymbol{\alpha}^{(i)},\kappa}(\mathbf{x})\phi_{\boldsymbol{\alpha}^{(j)},\kappa}(\mathbf{x})\right) \mathrm{d}\mathbf{x}. \tag{4.1}$$

While we will not discuss the exploitation of quadrature-free methods to compute the face integrals appearing in (3.2), we refer to [2] for the application of the proposed numerical integration approach for the computation of the face integrals arising from a DGFEM discretisation of a second-order elliptic problem.

4.1. Quadrature based assembly

As a point of comparison, we will consider quadrature rules of the form

$$\int_{\mathcal{P}} f(\mathbf{x})\,\mathrm{d}\mathbf{x} \approx \sum_{i=1}^{N} \omega_i f(\mathbf{x}_i), \tag{4.2}$$

where $f : \mathcal{P} \to \mathbb{R}$, $\{\mathbf{x}_i\}_{i=1}^N \subset \mathbb{R}^d$ denotes a set of quadrature points with non-negative quadrature weights $\{\omega_i\}_{i=1}^N \subset \mathbb{R}$.

A number of methods may be employed to construct the N-point quadrature scheme $Q_N = \{(\mathbf{x}_i, \omega_i)\}_{i=1}^N$. One of the most popular and simple to implement strategies is to sub-tessellate the integration domain $\mathcal{P}$ into simplicial subdomains (triangles in 2D, tetrahedra in 3D), on which standard quadrature schemes can be used [25]. However, the resulting quadrature scheme on $\mathcal{P}$ may contain an excessive number of points and weights. It has been demonstrated that numerical optimisation algorithms can generate efficient numerical quadrature schemes on arbitrary polygonal domains, cf., for example, [20].

In the case when $f \in \mathbb{P}^p(\mathcal{P})$, the approximation (4.2) can be exact. Indeed, it can be shown that the smallest N-point quadrature scheme which is exact for all polynomial functions of a given degree p contains

$$N \geq \binom{\lfloor \frac{p}{2} \rfloor + d}{d} \tag{4.3}$$

quadrature points and weights [23].

Algorithm 2 presents a pseudocode for a typical implementation of numerical quadrature to evaluate the integrals (4.1) appearing in the DGFEM discretisation of the transport equation. Noting that lines 8 and 9 of Algorithm 2 require $2(d+1)$ and 2 floating-point operations to evaluate, respectively, it can be seen that the number of floating-point operations performed in the main body (lines 5–12) is given by $2(d+2)(\dim \mathbb{P}^p(\kappa))^2 N$. Furthermore, noting that $\dim \mathbb{P}^p(\kappa) = \binom{p+d}{d}$ and that the integrand of (4.1) is a polynomial of total degree at most $2p$, the number of floating-point operations required to exactly evaluate $\mathbf{A}_\kappa$ for a single element κ using Algorithm 2 is at least $2(d+1)\binom{p+d}{d}^3$. Here, we have assumed that the lower bound on the number of quadrature points in (4.3) is attainable; that is, we set $N = N_{opt} = \binom{p+d}{d}$.

Algorithm 2 Computation of $\mathbf{A}_\kappa$ using quadrature.

```
 1: procedure ComputeElementMatrix($\kappa$)
 2:     Compute quadrature scheme $\{(\mathbf{x}_q, \omega_q)\}_{q=1}^N$ on $\kappa$
 3:     Pre-compute $\{\phi_{\alpha,\kappa}\}_{0\le|\alpha|\le p}$ and $\{\nabla\phi_{\alpha,\kappa}\}_{0\le|\alpha|\le p}$ at quadrature points
 4:     $\mathbf{A}_\kappa \leftarrow 0$
 5:     for $q = 1, \dots, N$ do
 6:         for $i = 1, \dots, \dim \mathbb{P}^p(\kappa)$ do
 7:             for $j = 1, \dots, \dim \mathbb{P}^p(\kappa)$ do
 8:                 $I \leftarrow \left(c\phi_{\alpha^{(i)},\kappa}(\mathbf{x}_q) - \mathbf{b}\cdot\nabla\phi_{\alpha^{(i)},\kappa}(\mathbf{x}_q)\right)\phi_{\alpha^{(j)},\kappa}(\mathbf{x}_q)$
 9:                 $(\mathbf{A}_\kappa)_{i,j} \leftarrow (\mathbf{A}_\kappa)_{i,j} + \omega_q I$
10:             end for
11:         end for
12:     end for
13:     return $\mathbf{A}_\kappa$
14: end procedure
```

4.2. Quadrature-free based assembly

The quadrature-free integration method outlined in Section 2 is not immediately applicable to the case of exactly evaluating the entries of $\mathbf{A}_\kappa$ in (4.1), since the integrand is typically not a homogeneous function. We remedy this issue by decomposing the integrand as a sum of monomials which may be integrated separately using Algorithm 1. We shall skip the details for brevity; see [2] for a more detailed treatment of similar integrals.

Since the basis functions $\{\phi_{\boldsymbol{\alpha},\kappa}\}_{0\le|\boldsymbol{\alpha}|\le p}$ are only supported on κ, we may apply the inverse map F_κ^{-1} to obtain an expression for the matrix entry $(\mathbf{A}_\kappa)_{i,j}$ as an integral over the mapped element

$$\hat{\kappa} = F_\kappa^{-1}(\kappa) \subseteq \hat{B}.$$

It can be shown that

$$(\mathbf{A}_\kappa)_{i,j} = \int_{\hat{\kappa}} \left(-\hat{\phi}_{\boldsymbol{\alpha}^{(j)}}(\hat{\mathbf{x}})\hat{\mathbf{b}}_\kappa \cdot \hat{\nabla}\hat{\phi}_{\boldsymbol{\alpha}^{(i)}}(\hat{\mathbf{x}}) + c\hat{\phi}_{\boldsymbol{\alpha}^{(i)}}(\hat{\mathbf{x}})\hat{\phi}_{\boldsymbol{\alpha}^{(j)}}(\hat{\mathbf{x}}) \right) |\mathbf{J}_\kappa| \,\mathrm{d}\hat{\mathbf{x}},$$

where $\hat{\mathbf{b}}_\kappa = \mathbf{J}_\kappa^{-1}\mathbf{b}$ denotes a scaled wind direction. Finally, by using the definition (3.3) of the basis functions $\{\hat{\phi}_{\boldsymbol{\alpha}}\}_{0\le\boldsymbol{\alpha}\le p}$ (which we remark are independent of κ) and the decompositions (3.4) and (3.5), we arrive at the following expression for $(\mathbf{A}_\kappa)_{i,j}$:

$$(\mathbf{A}_\kappa)_{i,j} = \sum_{\mathbf{0}\le\boldsymbol{\alpha}\le\boldsymbol{\alpha}^{(i)}+\boldsymbol{\alpha}^{(j)}} c_{\boldsymbol{\alpha}}^{(i,j)} \int_{\hat{\kappa}} \hat{\mathbf{x}}^{\boldsymbol{\alpha}} \,\mathrm{d}\hat{\mathbf{x}}, \tag{4.4}$$

where the coefficients $\{c_{\boldsymbol{\alpha}}^{(i,j)}\}_{\mathbf{0}\le\boldsymbol{\alpha}\le\boldsymbol{\alpha}^{(i)}+\boldsymbol{\alpha}^{(j)}}$ are defined for each $1 \le i, j \le \dim \mathbb{P}^p(\kappa)$ by

$$c_{\boldsymbol{\alpha}}^{(i,j)} = \left(c \prod_{k=1}^{d} C_{\alpha_k^{(i)},\alpha_k^{(j)},\alpha_k} - \sum_{k=1}^{d} \hat{b}_{\kappa,k} C'_{\alpha_k^{(i)},\alpha_k^{(j)},\alpha_k} \prod_{\substack{\ell=1 \\ \ell\ne k}}^{d} C_{\alpha_\ell^{(i)},\alpha_\ell^{(j)},\alpha_\ell} \right) |\mathbf{J}_\kappa|. \tag{4.5}$$

We remark that the entries of $\mathbf{A}_\kappa$ are now in a form in which Algorithm 1 can be applied to generate the set of integrated monomials $\mathcal{I}(\hat{\kappa}, \mathcal{J})$. Here, we make the choice

$$\mathcal{J} = \left\{ \boldsymbol{\alpha} \in \mathbb{N}_0^d : 0 \le |\boldsymbol{\alpha}| \le 2p \right\};$$

this ensures that $\mathcal{J}$ contains each $\boldsymbol{\alpha}$ for which $c_{\boldsymbol{\alpha}}^{(i,j)} \ne 0$. Moreover, by (4.4), the set $\mathcal{I}(\hat{\kappa}, \mathcal{J})$ can be computed once for each κ and re-used to assemble each entry of $\mathbf{A}_\kappa$.

Algorithm 3 provides pseudocode for a typical implementation of the quadrature-free based integration method to evaluate the integrals (4.1) appearing in the DGFEM discretisation of the transport equation. We remark that the computational complexity associated with the execution of Algorithm 1 on line 4 has already been discussed in Section 2.2. Noting that lines 10 and 11 of Algorithm 3 require $(d+1)^2$ and 2 floating-point operations, respectively, it can be seen that the number of floating-point operations performed in the main body (lines 7–14) is given by $((d+1)^2 + 2)Q_d(p)$, where

$$Q_d(p) = \sum_{0\le|\boldsymbol{\alpha}^{(i)}|\le p} \sum_{0\le|\boldsymbol{\alpha}^{(j)}|\le p} \mathrm{card}\{\boldsymbol{\alpha} : \mathbf{0} \le \boldsymbol{\alpha} \le \boldsymbol{\alpha}^{(i)} + \boldsymbol{\alpha}^{(j)}\}. \tag{4.6}$$

It can be shown that $Q_d(p) \sim \frac{1}{(3d)!}\binom{4d}{2d}p^{3d}$ as $p \to \infty$.

Algorithm 3 Computation of $\mathbf{A}_\kappa$ via quadrature-free integration.

```
1:  procedure ComputeElementMatrix($\kappa$)
2:      Compute sets of coefficients $\{C_{m,n,k}\}$ and $\{C'_{m,n,k}\}$ in (3.4) and (3.5) (if not already available)
3:      Map $\kappa \mapsto \hat{\kappa}$
4:      Compute $\mathcal{I}(\hat{\kappa}, \mathcal{J})$ using Algorithm 1
5:      Compute $\hat{\mathbf{b}}_\kappa = \mathbf{J}_\kappa^{-1}\mathbf{b}$
6:      $\mathbf{A}_\kappa \leftarrow 0$
7:      for $i = 1, \ldots, \dim \mathbb{P}^p(\kappa)$ do
8:          for $j = 1, \ldots, \dim \mathbb{P}^p(\kappa)$ do
9:              for $\mathbf{0} \leq \boldsymbol{\alpha} \leq \boldsymbol{\alpha}^{(i)} + \boldsymbol{\alpha}^{(j)}$ do
10:                 Compute $c_{\boldsymbol{\alpha}}^{(i,j)}$ as in (4.5)
11:                 $(\mathbf{A}_\kappa)_{i,j} \leftarrow (\mathbf{A}_\kappa)_{i,j} + c_{\boldsymbol{\alpha}}^{(i,j)}\mathcal{I}(\hat{\kappa}, \boldsymbol{\alpha})$
12:             end for
13:         end for
14:     end for
15:     return $\mathbf{A}_\kappa$
16: end procedure
```

Remark 6 (Pre-computation of coefficients). *One may optionally pre-compute the (κ-independent) coefficients*

$$C_{\boldsymbol{\alpha}^{(i)},\boldsymbol{\alpha}^{(j)},\boldsymbol{\alpha}} = \prod_{k=1}^{d} C_{\alpha_k^{(i)},\alpha_k^{(j)},\alpha_k}$$

and

$$C^{(k)}_{\boldsymbol{\alpha}^{(i)},\boldsymbol{\alpha}^{(j)},\boldsymbol{\alpha}} = C'_{\alpha_k^{(i)},\alpha_k^{(j)},\alpha_k} \prod_{\substack{\ell=1\\ \ell\neq k}}^{d} C_{\alpha_\ell^{(i)},\alpha_\ell^{(j)},\alpha_\ell}$$

for $0 \leq |\boldsymbol{\alpha}^{(i)}| \leq p$, $0 \leq |\boldsymbol{\alpha}^{(j)}| \leq p$, $\mathbf{0} \leq \boldsymbol{\alpha} \leq \boldsymbol{\alpha}^{(i)} + \boldsymbol{\alpha}^{(j)}$ *and* $1 \leq k \leq d$. *This allows (4.5) to be computed using* $2(d+1)$ *floating-point operations, which is the same as the number of floating-point operations needed to evaluate I in the quadrature based implementation given in Algorithm 2.*

Remark 7 (p-refinement in quadrature-free based assembly). *Algorithm 3 requires the following sets in order to assemble the matrix* $\mathbf{A}_\kappa$ *using a basis of* $\mathbb{P}^p(\kappa)$*:*

$$\begin{aligned} C(p) &= \left\{C_{i,j,k} : 0 \leq i \leq p, 0 \leq j \leq p, 0 \leq k \leq i+j\right\}, \\ C'(p) &= \left\{C'_{i,j,k} : 0 \leq i \leq p, 0 \leq j \leq p, 0 \leq k \leq i+j\right\}, \\ \mathcal{I}_\kappa(p) &= \{\mathcal{I}(\mathcal{F}, \boldsymbol{\alpha}) : 0 \leq \dim \mathcal{F} \leq d, 0 \leq |\boldsymbol{\alpha}| \leq 2p\}. \end{aligned}$$

Suppose we wish to perform a p-refinement; that is, to construct $\mathbf{A}_\kappa$ *using a basis of* $\mathbb{P}^{p+1}(\kappa)$*. Further, assume that* $C(p)$*,* $C'(p)$ *and* $\mathcal{I}_\kappa(p)$ *are already known. We have that*

$$C(p+1) = C(p) \cup \left\{C_{i,p,k} : 0 \leq i \leq p+1,\ 0 \leq k \leq p+i\right\}$$

$$
\begin{aligned}
&\cup \left\{C_{p,i,k} : 0 \le i \le p+1,\ 0 \le k \le p+i\right\},\\
C'(p+1) &= C'(p) \cup \left\{C'_{i,p,k} : 0 \le i \le p+1,\ 0 \le k \le p+i\right\}\\
&\cup \left\{C'_{p,i,k} : 0 \le i \le p+1,\ 0 \le k \le p+i\right\},\\
\mathcal{I}_\kappa(p+1) &= \mathcal{I}_\kappa(p) \cup \{\mathcal{I}(\mathcal{F},\boldsymbol{\alpha}) : 0 \le \dim\mathcal{F} \le d,\ 2p+1 \le |\boldsymbol{\alpha}| \le 2p+2\}.
\end{aligned}
$$

The computations of the updated coefficient sets $C(p+1)$ and $C'(p+1)$ are a one-time cost (since these sets may be used for every $\kappa \in \mathcal{T}$) and the computation of the updated integral set $\mathcal{I}_\kappa(p+1)$ can be performed using a modification of Algorithm 1 that uses prior knowledge of $\mathcal{I}_\kappa(p)$ to avoid unnecessary re-computation.

4.3. Comparison of assembly methods

It can be seen that the time complexities associated with the main loops in Algorithms 2 and 3, measured as the total number of floating-point operations required to assemble $\mathbf{A}_\kappa$, are $O(p^{3d})$ in the limit as $p \to \infty$. Moreover, the time complexity associated with the execution of Algorithm 1, used to compute $\mathcal{I}(\hat{\kappa},\mathcal{J})$ in Algorithm 3, is $O(\chi_1(\kappa)p^d)$, where $\chi_1(\mathcal{P})$ denotes the measure of complexity of the geometry of a polytope $\mathcal{P}$ given in the statement of Lemma 1. Thus, for large enough p, the main loops in Algorithms 2 and 3 are the most expensive contributions to the total assembly time.

A study of the loops in Algorithms 2 and 3 shows that the quadrature based method reaches the inner-most computations $(\dim\mathbb{P}^p(\kappa))^2N$ times, while the quadrature-free based method reaches the inner-most computations $Q_d(p)$ times, where $Q_d(p)$ is the function defined in (4.6). Here, the number of quadrature points and weights N used in Algorithm 2 is chosen to exactly evaluate the integrals $(\mathbf{A}_\kappa)_{ij}$ in (4.1) whose integrands are polynomial functions of total degree at most $2p$. Thus, a lower bound for N is given by $N_{opt} = \binom{p+d}{d}$.

Figure 3 shows the expected number of floating-point operations required by Algorithm 2 (using the theoretically-optimal number of quadrature points N_{opt}) and Algorithm 3 used to assemble $\mathbf{A}_\kappa$. The leading-order behaviour of both algorithms in the limit $p \to \infty$ is $O(p^{3d})$. It is seen that, without taking the geometric complexity of κ into consideration, the performance of the quadrature-free based assembly method is expected to be comparable to that of the quadrature based assembly employing the minimal quadrature set that exactly evaluates $\mathbf{A}_\kappa$.

However, the quadrature-free based assembly method can outperform the quadrature based assembly method when one takes into consideration the geometric complexity of κ. To see this, suppose that κ is decomposed into n subdomains for the purpose of numerical integration, on each of which an N_{opt}-point quadrature scheme can be applied. The leading-order behaviour of Algorithm 2 is $O(np^{3d})$; that is, the time taken to assemble $\mathbf{A}_\kappa$ via the quadrature based assembly method will increase significantly if many integration subdomains are required. In contrast, the time taken to execute the main loop of Algorithm 3 is independent of κ; furthermore, taking into account the evaluation of the set of integrated monomials $\mathcal{I}(\hat{\kappa},\mathcal{J})$, the leading-order behaviour of Algorithm 3 is $O(\chi_1(\hat{\kappa})p^d + p^{3d})$, where χ_1 denotes the function given in the statement of Lemma 1. We investigate this further in the following section.

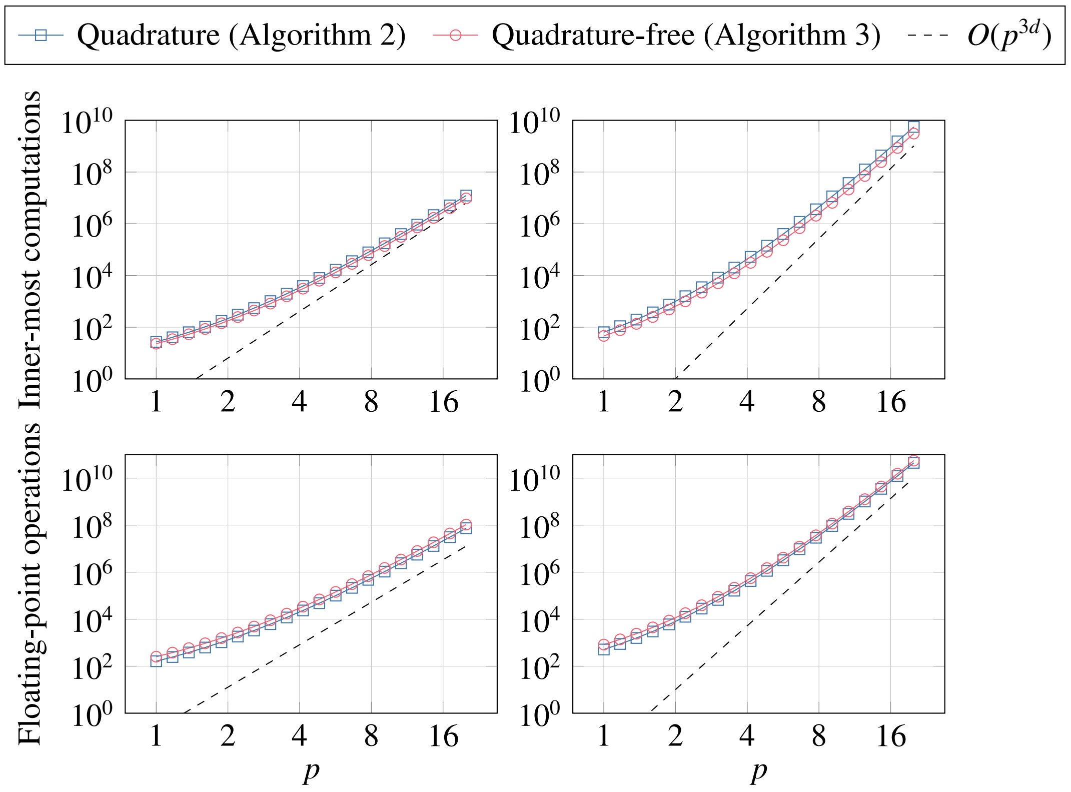


Figure 3. Time complexities of the main loops in Algorithms 2 and 3 as a function of the degree of approximation p. It is assumed that $N = N_{opt}$ quadrature points are used in Algorithm 2. Top row: number of times the operations within the main loops of Algorithms 2 and 3 are executed. Bottom row: total number of floating-point operations computed within the main loops of Algorithms 2 and 3. Left column: $d = 2$. Right column: $d = 3$.

5. Numerical results

In this section we consider the practical performance of the proposed quadrature-free algorithm.

5.1. Effect of pruning in Algorithm 1

We shall first study the effect of implementing Algorithm 1 with and without pruning as described in Section 2.2. The pruning strategy we adopt is to select the reference point $\mathbf{x}_{\mathcal{F}}$ as the first vertex of $\mathcal{F}$ for each face visited by Algorithm 1. We will apply Algorithm 1 to the problem of assembling the integral sets $\bigcup_{\kappa\in\mathcal{T}} \mathcal{I}(\kappa,\mathcal{J})$ in the case where $\mathcal{T}$ is a simplicial or agglomerated simplicial mesh in two or three spatial dimensions and

$$\mathcal{J} = \left\{\alpha \in \mathbb{N}_0^d : 0 \le |\alpha| \le p\right\}$$

for $p \in \{0, 2, 4, 6, 8, 10, 12\}$.

Tables 2 and 3 show the total CPU time taken by the unpruned and pruned versions of Algorithm 1 applied to two-dimensional triangular and agglomerated triangular meshes, respectively, while Tables 4 and 5 show the total CPU time taken by the unpruned and pruned versions of Algorithm 1 applied to three-dimensional tetrahedral and agglomerated tetrahedral meshes, respectively. An additional quantity, computed as the ratio of the CPU time taken by the pruned algorithm against the unpruned algorithm, is also reported; values of this ratio less than 1 indicate that Algorithm 1 with pruning computes the integral set $\mathcal{I}(\mathcal{P}, \mathcal{J})$ faster than the same algorithm without pruning.

For each fixed p, it is observed that the ratio of CPU time taken by the pruned algorithm against the unpruned algorithm remains roughly constant in all cases. For a fixed number of elements, this ratio decreases as p increases. It is expected that this ratio continues to decrease as $p \to \infty$ but remains bounded from below by a constant, for a single element κ, this constant is expected to depend on the number of nodes and edges of the graph $G(\kappa)$ defined in Section 2.2 before and after pruning.

While pruning accelerates the assembly of $\bigcup_{\kappa \in \mathcal{T}} \mathcal{I}(\kappa, \mathcal{J})$ for both tetrahedral and agglomerated tetrahedral elements in three dimensions, a greater improvement in assembly time is observed for tetrahedral meshes, this is because a greater proportion of the nodes and edges of $G(\kappa)$ can be eliminated through pruning when κ is simplicial. We observe similar behaviour between the pruned and unpruned version of Algorithm 1 applied to triangular and agglomerated triangular elements in two dimensions, though pruning is seen to be less effective at reducing the CPU time spent assembling $\bigcup_{\kappa \in \mathcal{T}} \mathcal{I}(\kappa, \mathcal{J})$. For small values of p, pruning actually slows down Algorithm 1, we speculate that this is because the extra time spent checking whether a given facet $\mathcal{F}$ is to be pruned outweighs the expense that would be incurred to assemble $\mathcal{I}(\mathcal{F}, \mathcal{J})$ without pruning.

Table 2. CPU times of unpruned and pruned versions of Algorithm 1 for the assembly of $\bigcup_{\kappa \in \mathcal{T}} \mathcal{I}(\kappa, \mathcal{J})$ for triangular meshes $\mathcal{T}$ in two spatial dimensions.

	$\lvert\mathcal{T}\rvert$	32	128	512	2048	8192
	Unpruned	1.8926E-05	6.2850E-05	2.3568E-04	8.8175E-04	3.6211E-03
$p = 0$	Pruned	2.4888E-05	7.3720E-05	2.7315E-04	1.0641E-03	4.5114E-03
	Ratio	1.32	1.17	1.16	1.21	1.25
	Unpruned	3.3402E-05	8.2150E-05	3.3563E-04	1.3032E-03	5.1413E-03
$p = 2$	Pruned	2.8993E-05	8.0898E-05	3.1069E-04	1.2043E-03	4.7641E-03
	Ratio	0.87	0.98	0.93	0.92	0.93
	Unpruned	3.3678E-05	1.2722E-04	5.1513E-04	1.7583E-03	6.4381E-03
$p = 4$	Pruned	3.1136E-05	9.4751E-05	3.6298E-04	1.5314E-03	5.5922E-03
	Ratio	0.92	0.74	0.70	0.87	0.87
	Unpruned	4.4647E-05	1.6264E-04	6.3257E-04	2.6062E-03	9.5336E-03
$p = 6$	Pruned	3.8436E-05	1.2308E-04	5.2076E-04	1.9159E-03	7.2723E-03
	Ratio	0.86	0.76	0.82	0.74	0.76
	Unpruned	6.4398E-05	2.2496E-04	9.3172E-04	3.2624E-03	1.2828E-02
$p = 8$	Pruned	4.5497E-05	1.7977E-04	7.8489E-04	2.5205E-03	9.4220E-03
	Ratio	0.71	0.80	0.84	0.77	0.73
	Unpruned	7.9538E-05	3.2234E-04	1.1563E-03	4.5004E-03	1.7702E-02
$p = 10$	Pruned	5.6676E-05	2.2770E-04	7.9667E-04	3.0367E-03	1.2133E-02
	Ratio	0.71	0.71	0.69	0.67	0.69
	Unpruned	1.3727E-04	4.1301E-04	1.6069E-03	5.7918E-03	2.3696E-02
$p = 12$	Pruned	6.9956E-05	2.7148E-04	9.6113E-04	3.7934E-03	1.5079E-02
	Ratio	0.51	0.66	0.60	0.65	0.64

Table 3. CPU times of unpruned and pruned versions of Algorithm 1 for the assembly of $\bigcup_{\kappa\in\mathcal{T}} \mathcal{I}(\kappa,\mathcal{J})$ for agglomerated triangular meshes $\mathcal{T}$ in two spatial dimensions.

	$\lvert\mathcal{T}\rvert$	12	51	204	819	3276
	Unpruned	2.3893E-05	6.7615E-05	2.7051E-04	1.1335E-03	5.0569E-03
$p = 0$	Pruned	3.1631E-05	8.4488E-05	3.1938E-04	1.3849E-03	6.2978E-03
	Ratio	1.32	1.25	1.18	1.22	1.25
	Unpruned	2.9957E-05	1.0833E-04	3.4219E-04	1.6801E-03	6.6029E-03
$p = 2$	Pruned	3.3662E-05	1.1878E-04	3.8369E-04	1.8276E-03	7.3026E-03
	Ratio	1.12	1.10	1.12	1.09	1.11
	Unpruned	4.6595E-05	1.4459E-04	5.9718E-04	2.2960E-03	9.4577E-03
$p = 4$	Pruned	3.9385E-05	1.2878E-04	5.9189E-04	2.2926E-03	9.2390E-03
	Ratio	0.85	0.89	0.99	1.00	0.98
	Unpruned	5.6086E-05	2.1821E-04	9.3389E-04	3.2810E-03	1.2832E-02
$p = 6$	Pruned	5.5423E-05	1.9397E-04	8.2830E-04	2.9239E-03	1.1699E-02
	Ratio	0.99	0.89	0.89	0.89	0.91
	Unpruned	8.5570E-05	3.0688E-04	1.1368E-03	4.5906E-03	1.7794E-02
$p = 8$	Pruned	8.1338E-05	2.5567E-04	9.8448E-04	3.8316E-03	1.5385E-02
	Ratio	0.95	0.83	0.87	0.83	0.86
	Unpruned	1.2774E-04	4.2015E-04	1.5151E-03	6.0202E-03	2.4115E-02
$p = 10$	Pruned	8.7737E-05	3.4614E-04	1.2627E-03	5.0212E-03	1.9915E-02
	Ratio	0.69	0.82	0.83	0.83	0.83
	Unpruned	1.5864E-04	5.3115E-04	2.0558E-03	8.2953E-03	3.3196E-02
$p = 12$	Pruned	1.4783E-04	4.3479E-04	1.7454E-03	6.7921E-03	2.5964E-02
	Ratio	0.93	0.82	0.85	0.82	0.78

Table 4. CPU times of unpruned and pruned versions of Algorithm 1 for the assembly of $\bigcup_{\kappa\in\mathcal{T}} \mathcal{I}(\kappa,\mathcal{J})$ for tetrahedral meshes $\mathcal{T}$ in three spatial dimensions.

	$\lvert\mathcal{T}\rvert$	6	48	384	3072	24576
	Unpruned	2.4926E-05	1.0816E-04	8.1476E-04	6.8287E-03	4.8521E-02
$p = 0$	Pruned	1.9337E-05	7.1661E-05	5.1822E-04	4.0497E-03	3.1205E-02
	Ratio	0.78	0.66	0.64	0.59	0.64
	Unpruned	3.5647E-05	2.0676E-04	1.5616E-03	1.3476E-02	9.5595E-02
$p = 2$	Pruned	2.1014E-05	9.8906E-05	7.1403E-04	5.6071E-03	4.3727E-02
	Ratio	0.59	0.48	0.46	0.42	0.46
	Unpruned	6.3102E-05	4.2202E-04	3.2991E-03	2.5008E-02	1.9898E-01
$p = 4$	Pruned	3.1368E-05	1.5550E-04	1.2420E-03	9.1364E-03	7.2012E-02
	Ratio	0.50	0.37	0.38	0.37	0.36
	Unpruned	1.1175E-04	8.2232E-04	5.9819E-03	4.7363E-02	3.7750E-01
$p = 6$	Pruned	4.4362E-05	2.5484E-04	1.8026E-03	1.3952E-02	1.1175E-01
	Ratio	0.40	0.31	0.30	0.29	0.30
	Unpruned	1.8889E-04	1.4290E-03	1.0750E-02	8.6211E-02	6.8734E-01
$p = 8$	Pruned	6.2170E-05	2.8795E-04	3.0248E-03	2.3477E-02	1.8765E-01
	Ratio	0.33	0.20	0.28	0.27	0.27

Table 5. CPU times of unpruned and pruned versions of Algorithm 1 for the assembly of $\bigcup_{\kappa\in\mathcal{T}} \mathcal{I}(\kappa,\mathcal{J})$ for agglomerated tetrahedral meshes $\mathcal{T}$ in three spatial dimensions.

	$\lvert\mathcal{T}\rvert$	4	38	307	2457	19660
	Unpruned	4.5272E-05	3.5574E-04	2.6056E-03	2.1919E-02	1.7550E-01
$p = 0$	Pruned	3.7088E-05	2.8895E-04	2.0201E-03	1.6717E-02	1.3413E-01
	Ratio	0.82	0.81	0.78	0.76	0.76
	Unpruned	8.7692E-05	7.7001E-04	5.5442E-03	4.3656E-02	3.5131E-01
$p = 2$	Pruned	5.4265E-05	4.5130E-04	3.1994E-03	2.6491E-02	2.0647E-01
	Ratio	0.62	0.59	0.58	0.61	0.59
	Unpruned	1.9519E-04	1.5893E-03	1.1656E-02	9.1985E-02	7.4515E-01
$p = 4$	Pruned	9.1346E-05	7.6676E-04	5.5029E-03	4.4103E-02	3.5562E-01
	Ratio	0.47	0.48	0.47	0.48	0.48
	Unpruned	3.4222E-04	3.0026E-03	2.3204E-02	1.8303E-01	1.4867E+00
$p = 6$	Pruned	1.6222E-04	1.2524E-03	9.9395E-03	7.9390E-02	6.4008E-01
	Ratio	0.47	0.42	0.43	0.43	0.43
	Unpruned	6.0944E-04	5.2648E-03	4.1087E-02	3.2613E-01	2.6380E+00
$p = 8$	Pruned	2.4756E-04	1.9114E-03	1.5453E-02	1.2295E-01	1.0089E+00
	Ratio	0.41	0.36	0.38	0.38	0.38

5.2. Integrating monomials over polygons

As a second example, we compare the quadrature and quadrature-free based integration algorithms for the evaluation of the sets

$$\mathcal{I}_{n,p} = \left\{ \int_{\mathcal{P}_n} \mathbf{x}^{\boldsymbol{\alpha}} \,\mathrm{d}\mathbf{x} : \boldsymbol{\alpha} \in \mathbb{N}_0^2, 0 \le |\boldsymbol{\alpha}| \le p \right\},$$

where $\mathcal{P}_n \subset \mathbb{R}^2$ denotes the regular n-gon, $5 \le n \le 16$, with vertices $\left\{\left(\cos\frac{2\pi k}{n}, \sin\frac{2\pi k}{n}\right)\right\}_{k=0}^{n-1}$ and $p \in \{2, 4, 8, 16, 32\}$.

The quadrature based method is employed as follows: A sub-tessellation of $\mathcal{P}_n$ consisting of $(n-2)$ triangles is constructed by joining the first vertex of $\mathcal{P}_n$ to every other vertex. On each triangle, a $(q + 1)^2$-point quadrature scheme, $q = \lceil\frac{p+1}{2}\rceil$, is defined by constructing a quadrature scheme on the unit square $(-1, 1)^2$ exactly integrating all bivariate polynomial functions of maximal degree $p + 1$. The reference quadrature scheme on $(-1, 1)^2$ is then mapped to each triangle in the sub-tessellation of $\mathcal{P}_n$ via a Duffy transformation [11]. The resulting quadrature scheme on $\mathcal{P}_n$ therefore contains $(n - 2)(q + 1)^2$ points and weights. We record the time taken for the quadrature based integration method to be executed for each element of $\mathcal{I}_{n,p}$; here, we do not include the time taken to generate the quadrature scheme on $\mathcal{P}_n$.

The quadrature-free based method is specialised to the two-dimensional setting. The integrals $\mathcal{I}(\mathcal{P}_n, \boldsymbol{\alpha})$ and $\mathcal{I}(\mathcal{F}_{n,k}, \boldsymbol{\alpha})$, for each boundary facet $\mathcal{F}_{n,k} \subset \partial\mathcal{P}_n$, $1 \le k \le n$, are stored in two arrays. We employ pruning based on selecting $\mathbf{x}_{\mathcal{F}}$ as the first vertex of each visited face $\mathcal{F}$.

Figures 4 and 5 show the CPU times taken by the quadrature based and quadrature-free based integration algorithms to evaluate $\mathcal{I}_{n,p}$ averaged over 100 function calls. The time complexity of the quadrature based method is $O(np^4)$, since the size of the requested set of integrals $|\mathcal{I}_{n,p}| = O(p^2)$ and each integral requires $O(n(q + 1)^2) = O(np^2)$ flops to compute. On the other hand, the quadrature-free based method is seen to have time complexity $O(np^2)$. This is consistent with Lemma 2 since $|J| = O(p^2)$. It has been verified that the integrals in the set $\mathcal{I}_{n,p}$ computed using Algorithm 1 agree with the same set of integrals computed using quadrature to within machine precision.

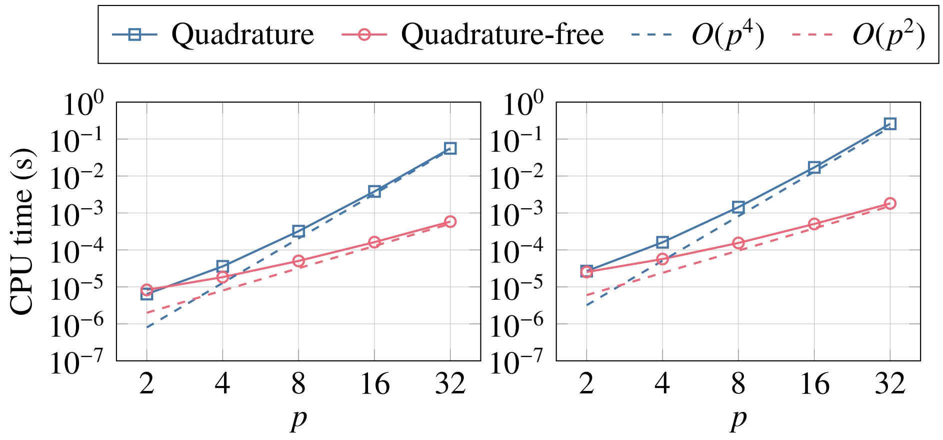


Figure 4. CPU times taken by the quadrature and quadrature-free based methods to evaluate $\mathcal{I}_{n,p}$ for $p = 2, 4, 8, 16, 32$ on a regular n-gon. Left: $n = 5$. Right: $n = 16$.

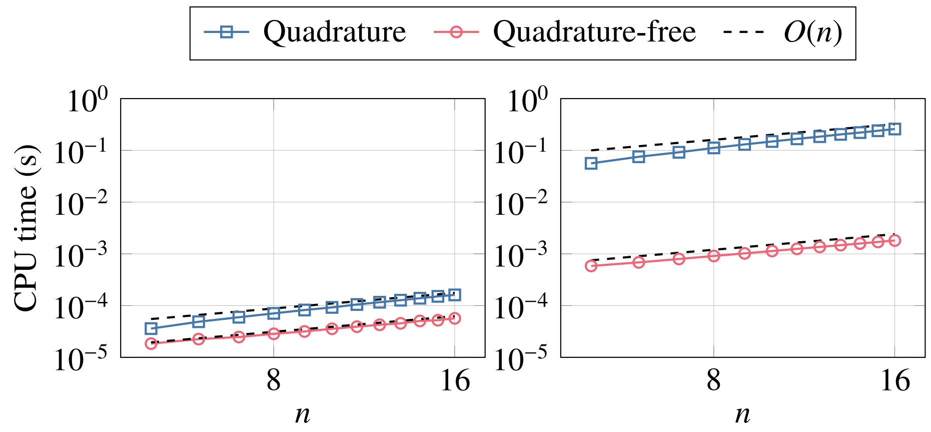


Figure 5. CPU times taken by the quadrature and quadrature-free based methods to evaluate $\mathcal{I}_{n,p}$ for $5 \le n \le 16$ and fixed p. Left: $p = 4$. Right: $p = 32$.

5.3. 2D transport matrix assembly

As a third example, we compare the quadrature based and quadrature-free based assembly methods for a single element matrix arising from the DGFEM discretisation of linear first-order transport problems posed in two spatial dimensions. We first consider a single n-sided polygonal domain $\mathcal{P}_n \subset \mathbb{R}^2$ with $3 \le n \le 64$ and employ a basis of degree $p = 4$, i.e., the finite element space is $\mathbb{V} = \mathbb{P}^4(\mathcal{P}_n)$.

As before, the quadrature based method employs a sub-tessellation for the purposes of constructing a quadrature scheme on $\mathcal{P}_n$. In this example, the sub-tessellation consists of n triangles constructed by joining each vertex of $\mathcal{P}_n$ to the centroid. Here, a $(p + 2)^2$-point quadrature scheme is employed on each triangle using the method outlined in the previous example; this ensures that the quadrature scheme exactly evaluates the element integrals appearing in (3.2). As before, we do not include the time taken to generate the quadrature scheme on $\mathcal{P}_n$. The time taken to evaluate the basis functions at the quadrature points is also excluded.

The quadrature-free based method is specialised to the two-dimensional setting. The element integrals appearing in (3.2) are evaluated using the two-step procedure given in Algorithm 3. The monomial integrals $\int_{\mathcal{P}_n} \mathbf{x}^{\alpha}\,\mathrm{d}\mathbf{x}$ for $0 \leq |\alpha| \leq 2p$ are computed once based on employing Algorithm 1 using the first-vertex based pruning strategy as before; these are then used to assemble the local element matrix entry-wise through the decomposition (4.4) of the integrals given in (3.2). The time taken to generate the coefficients in these decompositions is not included, since for the given finite element basis, the individual terms present in the brackets in (4.5) can be precomputed once and for all, using, for example, `maple`.

Figure 6 shows the CPU time taken by the quadrature based assembly method using Algorithm 2 and the quadrature-free based assembly method using Algorithm 3 to assemble the element matrices arising from a DGFEM discretisation of a linear, constant-coefficient transport problem in two spatial dimensions. The CPU times are averaged over 10000 calls to both assembly methods on the same n-gon element $\mathcal{P}_n$ for $3 \leq n \leq 64$ on which a basis of $\mathbb{P}^4(\mathcal{P}_n)$ is employed. The time taken by Algorithm 3 to assemble the element matrices is further broken down into contributions from line 4 (i.e., Algorithm 1) and lines 7–14 (i.e., the reconstruction of $(\mathbf{A}_\kappa)_{i,j}$ via (4.4)).

As seen in the previous example, the time-complexity of Algorithm 2 scales linearly with n, the number of sides of the element $\mathcal{P}_n$. The time-complexity associated with line 4 of Algorithm 3, which we remark is a call to Algorithm 1 also scales linearly with n, as expected.

The main body of Algorithm 3, namely the nested loops on lines 7–14, is seen to scale independently of n. Indeed, it was seen in the analysis of the previous section that this loop exhibits no dependence on the geometry of the element. Therefore, the actual run-time of the quadrature-free based assembly method depends on whether the integration phase (line 4) or reconstruction phase (lines 7–14) is more expensive. For small values of n, the reconstruction phase is more expensive and so the assembly time remains roughly constant; for large enough n the integration phase dominates the computational time and so a dependence of the assembly time on the geometric complexity of the element emerges.

We now consider fixing n to study the dependence on p; to this end, in Figure 7 we show the scalings of Algorithms 2 and 3 as functions of the polynomial degree of approximation for $1 \leq p \leq 12$ on a fixed 6-sided polygonal domain. For large enough p, it can be seen that the time complexities of the quadrature based and quadrature-free based assembly methods are both $O(p^{3d}) = O(p^6)$ as predicted earlier. In the case of the quadrature-free based assembly method, the time complexities (on a given geometry) of the monomial integration phase (line 4) and the reconstruction phase (lines 7–14) of Algorithm 3 are $O(p^{3d}) = O(p^6)$ and $O(p^d) = O(p^2)$, respectively. This is in agreement with our previous analysis, which predicted that the time complexity of the quadrature-free based assembly scales like $O(\chi_1(\mathcal{P}_n)p^d + p^{3d}) = O(\chi_1(\mathcal{P}_n)p^2 + p^6)$, where $\chi_1(\mathcal{P})$ denotes the measure of complexity of the geometry of a polytope $\mathcal{P}$ given in the statement of Lemma 1. In this example, it is observed that the quadrature-free based assembly method is faster than the quadrature based assembly method by a factor of around an order of magnitude for all tested polynomial degrees.

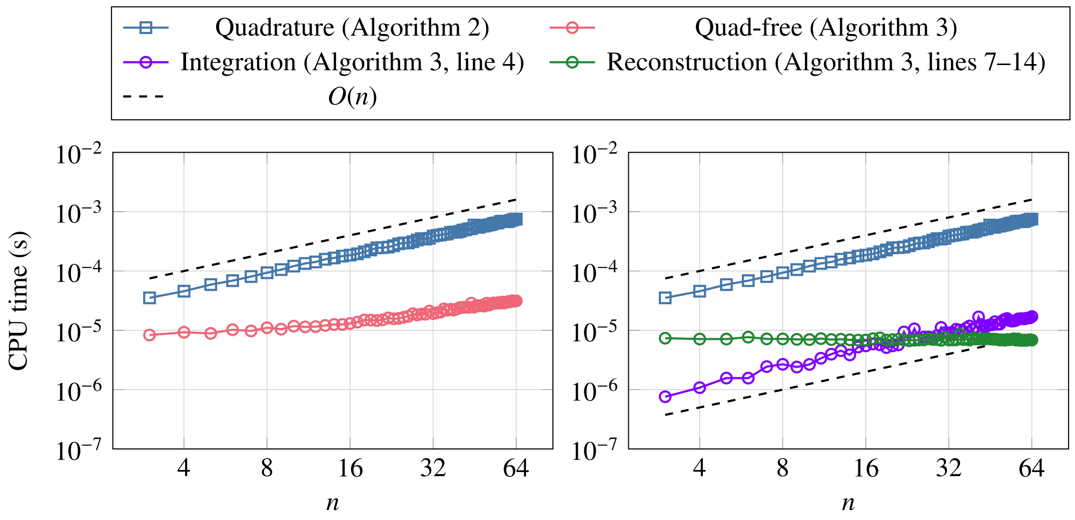


Figure 6. CPU times taken by the quadrature and quadrature-free based methods to evaluate the DGFEM element transport matrix for $3 \leq n \leq 64$ and fixed $p = 4$. Left: total time taken by quadrature and quadrature-free based methods. Right: contributions to CPU time arising from Algorithm 2 (blue), line 4 of Algorithm 3 (purple) and lines 7–14 of Algorithm 3 (green).

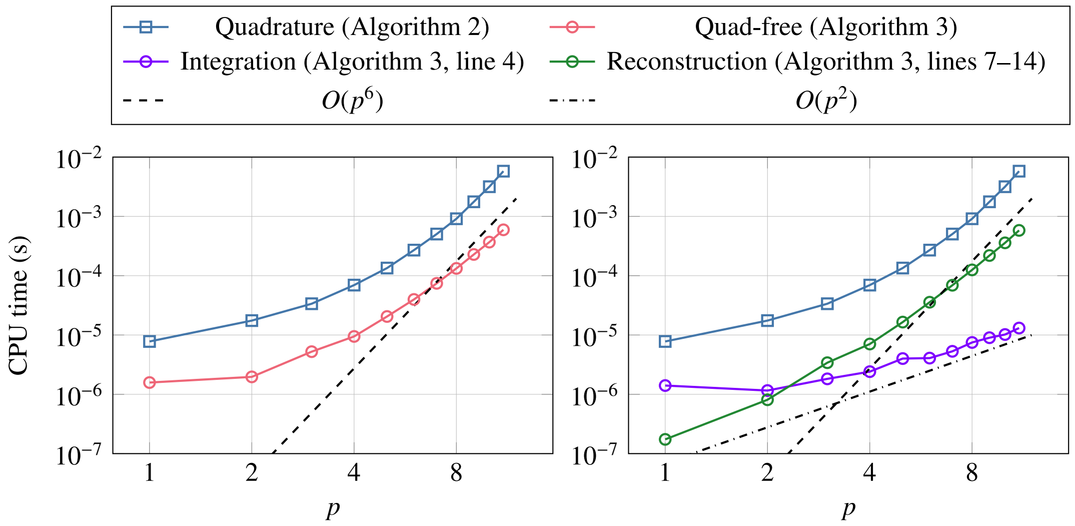


Figure 7. CPU times taken by the quadrature and quadrature-free based methods to evaluate the DGFEM element transport matrix for fixed $n = 6$ and $1 \leq p \leq 12$. Left: total time taken by quadrature and quadrature-free based methods. Right: Contributions to CPU time arising from Algorithm 2 (blue), line 4 of Algorithm 3 (purple) and lines 7–14 of Algorithm 3 (green).

5.4. 3D transport matrix assembly

As a final example, we will compare the quadrature based and quadrature-free based assembly methods for the system matrix arising from DGFEM discretisation of the linear first-order transport problem posed in three spatial dimensions. We will consider sequences of tetrahedral and agglomerated tetrahedral meshes $\mathcal{T}$ and employ local polynomial bases $\mathbb{P}^p(\kappa)$ for each $\kappa \in \mathcal{T}$ with $0 \leq p \leq 4$.

For standard tetrahedral meshes, the quadrature based method employs quadrature schemes consisting of $(p+2)^3$ points and weights on each tetrahedron; this ensures that the quadrature scheme exactly evaluates the element integrals appearing in (3.2). The agglomerated tetrahedral meshes are formed by partitioning a fine mesh $\mathcal{T}_{fine}$ into polyhedral coarse-mesh elements $\kappa \in \mathcal{T}$ using METIS [15]. The agglomeration strategy is chosen such that each coarse-mesh element κ is formed from an average of 10 fine-mesh elements $\kappa_{fine} \in \mathcal{T}_{fine}$. The quadrature schemes on elements in $\mathcal{T}_{fine}$ are inherited by the coarse-mesh elements, ensuring that the integrals appearing in (3.2) can be evaluated exactly.

The quadrature-free based method is performed in two steps as before. On each element $\kappa \in \mathcal{T}$, the monomial integrals $\int_\kappa \mathbf{x}^\alpha \, d\mathbf{x}$ for $0 \leq |\alpha| \leq 2p$ are computed once using an implementation of Algorithm 1 specialised to the three-dimensional setting. As before, we employ a first-vertex based pruning strategy to reduce the CPU time spent in Algorithm 1. The integrals in (3.2) are then evaluated using known decompositions of the integrands in terms of the monomial basis. The time taken to generate the coefficients in these decompositions is not included.

Figure 8 shows the CPU time taken by the quadrature based and quadrature-free based methods (Algorithms 2 and 3, respectively) to assemble the global transport matrices arising from a DGFEM discretisation of a linear, constant-coefficient transport problem in three spatial dimensions. Both methods are tested on standard and agglomerated tetrahedral meshes for global polynomial degrees $0 \leq p \leq 4$.

Both assembly methods are seen to scale linearly with the number of elements in the spatial mesh, as expected. For all tests recorded, the CPU time taken to assemble the system matrix using the quadrature-free method is consistently faster than the standard quadrature based approach by at most a constant multiplicative factor. For the tests performed on standard tetrahedral meshes, this multiplicative constant is between 2 and 3 for $p \geq 1$. For agglomerated tetrahedral meshes, this multiplicative constant improves to at least 5 for $p \geq 1$, with the quadrature based assembly taking almost 20 times longer than the quadrature-free based assembly in the case $p = 4$. This improvement in assembly time due to switching to a quadrature-free approach is expected to be greater on agglomerated meshes than tetrahedral meshes. Indeed, on a given element $\kappa \in \mathcal{T}$, the time complexity of the quadrature based method is $O(n_\kappa p^{3d}) = O(n_\kappa p^9)$, where n_κ denotes the number of fine-mesh elements in $\mathcal{T}_{fine}$ that comprise κ. In contrast, the time complexity of the quadrature-free based method is $O(\chi_1(\kappa)p^d + p^{3d}) = O(\chi_1(\kappa)p^3 + p^9)$.

Figures 9 and 10 present the breakdown of the contribution of line 4 and lines 7–14 of Algorithm 3 to the total CPU time taken by the quadrature-free based algorithm. Both the integration and reconstruction phases are seen to scale linearly with the number of elements in the mesh; this is to be expected since lines 4 and lines 7–14 of Algorithm 3 are executed once for each element. The reconstruction (4.4) performed on lines 7–14 of Algorithm 3 is seen to scale much faster as a function of p than the integration of the monomial basis via Algorithm 1; this is evidenced by the greater separation of the data corresponding to the CPU times for each polynomial degree p. However, the time taken in the reconstruction phase is seen to be independent of the geometry of the mesh elements, whereas a greater amount of CPU time is required to perform the integration phase on agglomerated tetrahedral meshes. Analogous behaviour is also observed when fine meshes consisting of hexahedral elements are agglomerated; for brevity these results have been omitted.

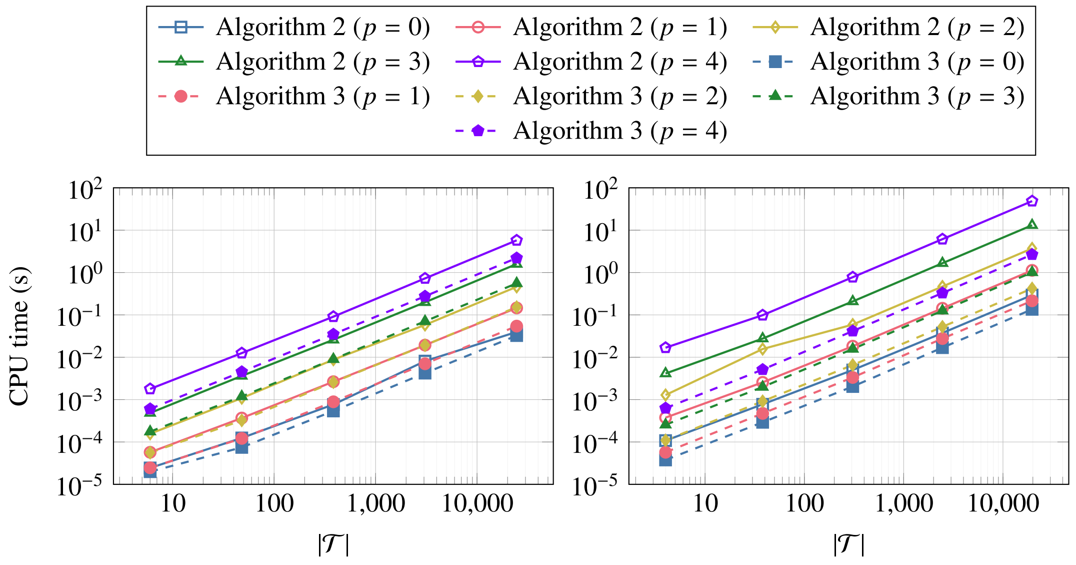


Figure 8. CPU times taken by Algorithm 2 (quadrature based assembly) and Algorithm 3 (quadrature-free based assembly) to evaluate the DGFEM element transport matrices on a sequence of meshes and for $0 \le p \le 4$. Left: total time taken on a sequence of tetrahedral meshes with $|\mathcal{T}| \in \{6, 48, 384, 3072, 24576\}$. Right: total time taken on a sequence of agglomerated meshes with $|\mathcal{T}| \in \{4, 38, 307, 2457, 19660\}$.

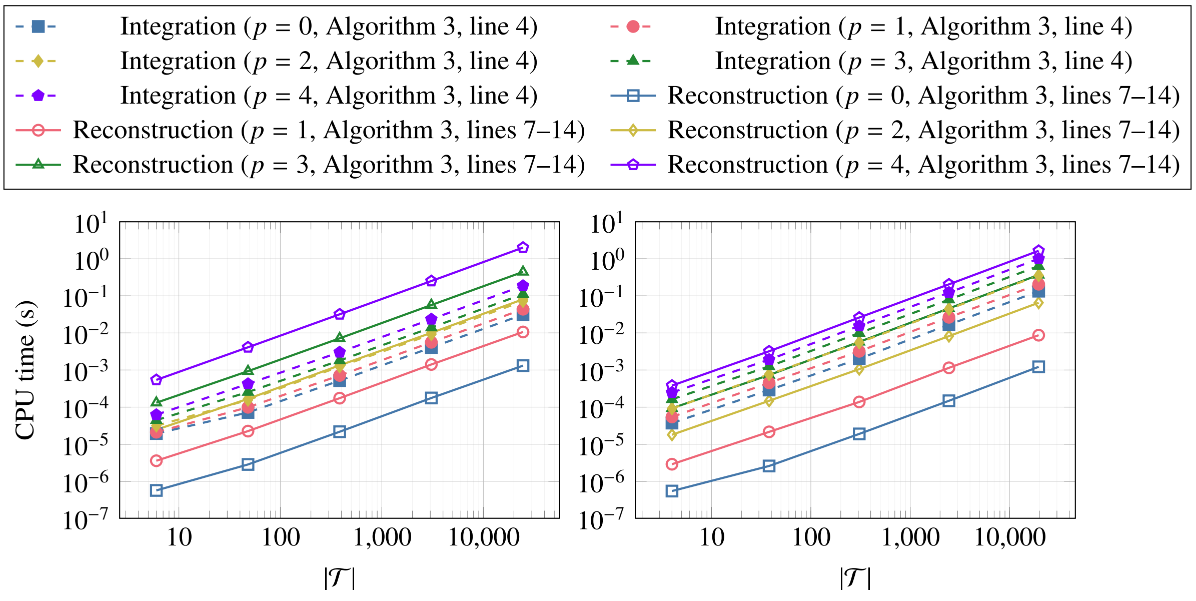


Figure 9. Breakdown of CPU times taken by the monomial integration and reconstruction phases of the quadrature-free based method to evaluate the DGFEM element transport matrices on a sequence of meshes and for $0 \le p \le 4$. Left: total time taken on a sequence of tetrahedral meshes with $|\mathcal{T}| \in \{6, 48, 384, 3072, 24576\}$. Right: total time taken on a sequence of agglomerated meshes with $|\mathcal{T}| \in \{4, 38, 307, 2457, 19660\}$.

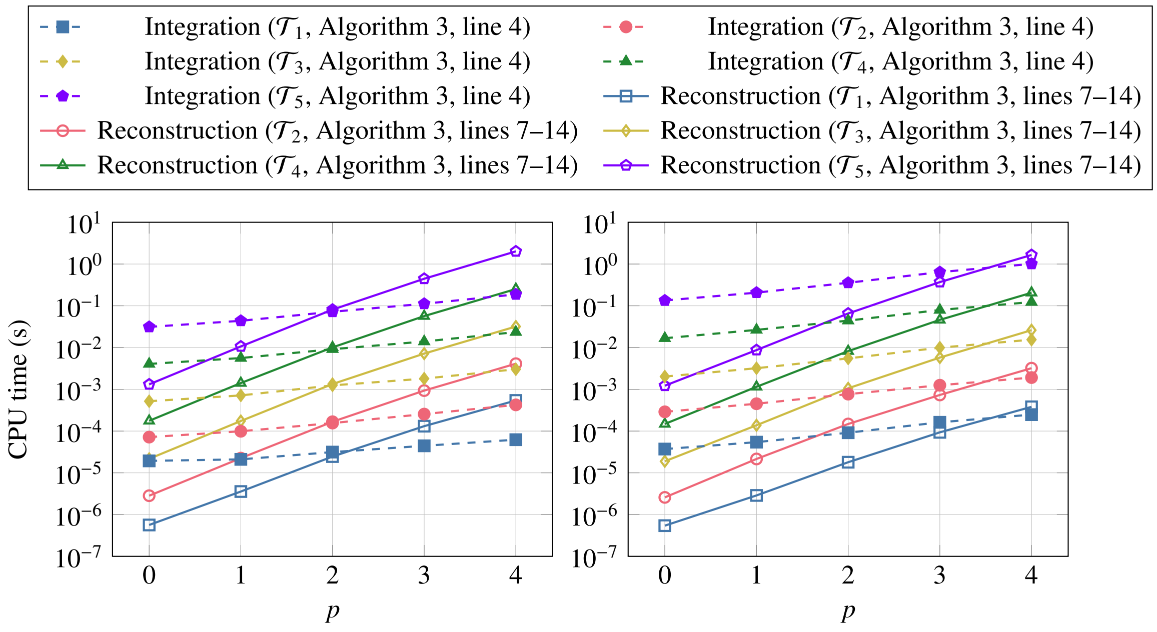


Figure 10. Breakdown of CPU times taken by the monomial integration and reconstruction phases of the quadrature-free based method to evaluate the DGFEM element transport matrices on sequences of meshes $(\mathcal{T}_i)_{i=1}^5$ and for $0 \leq p \leq 4$. Left: total time taken on a sequence of tetrahedral meshes with $|\mathcal{T}_i| \in \{6, 48, 384, 3072, 24576\}$. Right: total time taken on a sequence of agglomerated meshes with $|\mathcal{T}_i| \in \{4, 38, 307, 2457, 19660\}$.

6. Conclusions

In this article we have analysed the computational complexity of computing the integral of families of polynomial spaces over general polytopic domains. Starting from the ideas developed in [7, 17] for the integration of homogeneous functions, we have demonstrated that the time and space complexities required to integrate families of monomial functions are dependent on three factors: the ambient dimension of the polytopic domain; the size of the requested set of monomial integrals; and the size of a directed graph related to the polytopic domain. In the case of polygonal or polyhedral geometries, the monomial integration algorithm is shown to scale linearly with the number of graph edges. This algorithm was applied to the computation of element integrals arising in the DGFEM discretisation of the linear transport problem. We have shown that, by decomposing the integrand into a linear combination of monomial functions, these integrals could be evaluated at speeds comparable to methods based on employing quadrature schemes with a minimal number of points and weights. In comparison to quadrature based methods employing a sub-tessellation, the quadrature-free approach is seen to both significantly accelerate the assembly of the system matrix and also scale independently of the element geometry for sufficiently-high polynomial degrees.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

PH and MEH acknowledge the financial support of the EPSRC (grant EP/R030707/1). PH also acknowledges the financial support of the MRC (grant MR/T017988/1).

Conflict of interest

The authors declare no conflicts of interest.

References

1. P. F. Antonietti, A. Cangiani, J. Collis, Z. Dong, E. H. Georgoulis, S. Giani, et al., Review of discontinuous Galerkin finite element methods for partial differential equations on complicated domains, In: G. R. Barrenechea, F. Brezzi, A. Cangiani, E. H. Georgoulis, *Building bridges: connections and challenges in modern approaches to numerical partial differential equations*, Cham: Springer, **114** (2016), 281–310. https://doi.org/10.1007/978-3-319-41640-3_9
2. P. F. Antonietti, P. Houston, G. Pennesi, Fast numerical integration on polytopic meshes with applications to discontinuous Galerkin finite element methods, *J. Sci. Comput.*, **77** (2018), 1339–1370. https://doi.org/10.1007/s10915-018-0802-y
3. L. Beirão Da Veiga, F. Brezzi, A. Cangiani, G. Manzini, L. D. Marini, A. Russo, Basic principles of virtual element methods, *Math. Models Methods Appl. Sci.*, **23** (2013), 199–214. https://doi.org/10.1142/S0218202512500492
4. B. Büeler, A. Enge, K. Fukuda, Exact volume computation for polytopes: a practical study, In: G. Kalai, G. M. Ziegler, *Polytopes–Combinatorics and computation*, DMV Seminar, Basel: Birkhäuser, **29** (2000), 131–154. https://doi.org/10.1007/978-3-0348-8438-9_6
5. A. Cangiani, Z. Dong, E. H. Georgoulis, P. Houston, *hp-version discontinuous Galerkin methods on polygonal and polyhedral meshes*, Cham: Springer, 2017. https://doi.org/10.1007/978-3-319-67673-9
6. A. Cangiani, E. H. Georgoulis, P. Houston, hp-version discontinuous Galerkin methods on polygonal and polyhedral meshes, *Math. Models Methods Appl. Sci.*, **24** (2014), 2009–2041. https://doi.org/10.1142/S0218202514500146
7. E. B. Chin, J. B. Lasserre, N. Sukumar, Numerical integration of homogeneous functions on convex and nonconvex polygons and polyhedra, *Comp. Mech.*, **56** (2015), 967–981. https://doi.org/10.1007/s00466-015-1213-7
8. E. B. Chin, N. Sukumar, An efficient method to integrate polynomials over polytopes and curved solids, *Comput. Aided Geom. Design*, **82** (2020), 101914. https://doi.org/10.1016/j.cagd.2020.101914
9. M. Cicuttin, A. Ern, N. Pignet, *Hybrid high-order methods: a primer with applications to solid mechanics*, Cham: Springer, 2021. https://doi.org/10.1007/978-3-030-81477-9
10. Z. Dong, E. H. Georgoulis, T. Kappas, GPU-accelerated discontinuous Galerkin methods on polytopic meshes, *SIAM J. Sci. Comput.*, **43** (2021), C312–C334. https://doi.org/10.1137/20M1350984

11. M. G. Duffy, Quadrature over a pyramid or cube of integrands with a singularity at a vertex, *SIAM J. Numer. Anal.*, **19** (1982), 1260–1262. https://doi.org/10.1137/0719090
12. B. Grünbaum, V. Klee, M. A. Perles, G. C. Shephard, *Convex polytopes*, Vol. 16, 1 Ed., New York: Interscience, 1967.
13. P. Houston, M. E. Hubbard, T. J. Radley, O. J. Sutton, R. S. J. Widdowson, Efficient high-order space-angle-energy polytopic discontinuous Galerkin finite element methods for linear Boltzmann transport, *arXiv*, 2023. https://doi.org/10.48550/arXiv.2304.09592
14. V. Kaibel, M. E. Pfetsch, Computing the face lattice of a polytope from its vertex-facet incidences, *Comp. Geom.*, **23** (2002), 281–290. https://doi.org/10.1016/S0925-7721(02)00103-7
15. G. Karypis, V. Kumar, A fast and highly quality multilevel scheme for partitioning irregular graphs, *SIAM J. Sci. Comput.*, **20** (1998), 359–392. https://doi.org/10.1137/S1064827595287997
16. D. E. Knuth, *The art of computer programming*, Vol. 2, Seminumerical Algorithms, 3 Eds., Addison-Wesley, 1981.
17. J. Lasserre, Integration on a convex polytope, *Proc. Amer. Math. Soc.*, **126** (1998), 2433–2441.
18. J. B. Lasserre, Integration and homogeneous functions, *Proc. Amer. Math. Soc.*, **127** (1999), 813–818.
19. J. N. Lyness, G. Monegato, Quadrature rules for regions having regular hexagonal symmetry, *SIAM J. Numer. Anal.*, **14** (1977), 283–295. https://doi.org/10.1137/0714018
20. S. E. Mousavi, H. Xiao, N. Sukumar, Generalized Gaussian quadrature rules on arbitrary polygons, *Int. J. Numer. Methods Eng.*, **82** (2010), 99–113. https://doi.org/10.1002/nme.2759
21. S. E. Mousavi, N. Sukumar, Numerical integration of polynomials and discontinuous functions on irregular convex polygons and polyhedrons, *Comput. Mech.*, **47** (2011), 535–554. https://doi.org/10.1007/s00466-010-0562-5
22. C. P. Simon, L. Blume, *Mathematics for economists*, Vol. 7, New York: Norton, 1994.
23. A. Stroud, *Approximate calculation of multiple integrals*, Prentice-Hall series in automatic computation, Prentice-Hall, Inc., 1971.
24. Y. Sudhakar, W. A. Wall, Quadrature schemes for arbitrary convex/concave volumes and integration of weak form in enriched partition of unity methods, *Comput. Methods Appl. Mech. Eng.*, **258** (2013), 39–54. https://doi.org/10.1016/j.cma.2013.01.007
25. N. Sukumar, A. Tabarraei, Conforming polygonal finite elements, *Int. J. Numer. Methods Eng.*, **61** (2004), 2045–2066. https://doi.org/10.1002/nme.1141
26. M. E. Taylor, *Partial differential equations: basic theory*, Vol. 1, Springer, 1996.
27. H. Xiao, Z. Gimbutas, A numerical algorithm for the construction of efficient quadrature rules in two and higher dimensions, *Comput. Math. Appl.*, **59** (2010), 663–676. https://doi.org/10.1016/j.camwa.2009.10.027