High-order quadrature on multi-component domains implicitly defined by multivariate polynomials

Robert I. Saye

Mathematics Group, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, United States of America

A R T I C L E   I N F O

A B S T R A C T

A high-order quadrature algorithm is presented for computing integrals over curved surfaces and volumes whose geometry is implicitly defined by the level sets of (one or more) multivariate polynomials. The algorithm recasts the implicitly defined geometry as the graph of an implicitly defined, multi-valued height function, and applies a dimension reduction approach needing only one-dimensional quadrature. In particular, we explore the use of Gauss-Legendre and tanh-sinh methods and demonstrate that the quadrature algorithm inherits their high-order convergence rates. Under the action of h-refinement with q fixed, the quadrature schemes yield an order of accuracy of $2q$, where q is the one-dimensional node count; numerical experiments demonstrate up to 22nd order. Under the action of q-refinement with the geometry fixed, the convergence is approximately exponential, i.e., doubling q approximately doubles the number of accurate digits of the computed integral. Complex geometry is automatically handled by the algorithm, including, e.g., multi-component domains, tunnels, and junctions arising from multiple polynomial level sets, as well as self-intersections, cusps, and other kinds of singularities. A variety of numerical experiments demonstrates the quadrature algorithm on two- and three-dimensional problems, including: randomly generated geometry involving multiple high-curvature pieces; challenging examples involving high degree singularities such as cusps; adaptation to simplex constraint cells in addition to hyperrectangular constraint cells; and boolean operations to compute integrals on overlapping domains.

© 2021 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

1. Introduction

In this paper, we develop high-order quadrature methods for curved surfaces and volumes whose geometry is implicitly defined by the level sets of (one or more) multivariate polynomials. In particular, given a set of polynomial functions $\phi_i : \mathbb{R}^d \to \mathbb{R}$, $i = 1, \ldots, n$, we consider volume and surface integrals of the form

$$\int_{U \setminus \Gamma} f \, dx \quad \text{and} \quad \int_{U \cap \Gamma} g \, dS, \tag{1}$$

where the implicitly defined interface, $\Gamma := \bigcup_{i=1}^{n} \{\phi_i = 0\}$, is the union of each polynomial's zero level set (its "interface"), and $U \subset \mathbb{R}^d$ is a given hyperrectangle (e.g., a rectangle in two dimensions, or rectangular box in three dimensions). Here, the

integrands f and g need not be smooth across the entire hyperrectangle U; we instead assume that f and g are sufficiently smooth on each connected component of $U \setminus \Gamma$ and $U \cap \Gamma$, respectively. As such, our main problem formulation concerns a hyperrectangular domain which has been divided into one or more pieces by the zero level set of each polynomial, and we develop quadrature algorithms which can accurately compute integrals on each individual piece.

Integrals of the form (1) arise in a variety of applications involving implicitly defined geometry, including, e.g.: discretization methods for solving partial differential equations on domains with curved geometry, such as cut-cell, embedded boundary, immersed interface, and implicit mesh methods [48,49,30,37,11,9]; remapping in arbitrary Lagrangian-Eulerian methods [1]; multi-mesh methods, which employ overlapping meshes of different characteristics for different parts of a multi-physics problem [29]; and in the treatment of jump conditions and singular source terms [52,44,8,14]. In many such applications, the interfacial geometry is not known ahead of time and is instead computed dynamically, e.g., through the level set method [42,56,41,57]. It is often important to compute these integrals to a high degree of accuracy, e.g., to ensure consistency in a weak formulation or to sufficiently diminish the impact of variational errors. Further, the implicitly defined geometry can be complex, even in a single grid cell, having characteristics such as tunnels, junctions, corners, or cusps; in these cases, it would be ideal to directly treat the geometry at hand, without having to resort to heuristic subdivision/refinement algorithms.

The quadrature algorithms developed in this work are designed with these aspects in mind and have the following features:

- The output is a quadrature scheme of the form

$$\int_{U \setminus \Gamma} f \approx \sum_i w_i f(x_i) \quad \text{and} \quad \int_{U \cap \Gamma} g \approx \sum_j \mathsf{w}_j g(\mathsf{x}_j),$$

  where the weights w_i, w_j are strictly positive and the nodes x_i, x_j are strictly inside their respective domains. Importantly, the quadrature schemes yield accurate methods for computing integrals on the individual connected components of these domains, e.g., if one polynomial ϕ implicitly defines Γ, then $\sum_{i|\phi(x_i)<0} w_i f(x_i)$ constitutes an accurate quadrature scheme for $\int_{U \cap \{\phi<0\}} f$, and there is no need to define or extend f throughout all of U.
- The algorithm employs two kinds of one-dimensional quadrature, Gauss-Legendre and tanh-sinh, and inherits their high-order convergence rates. In d dimensions, based on a one-dimensional scheme of order q, the total number of quadrature nodes is $\mathcal{O}(q^d)$ in the volumetric case and $\mathcal{O}(q^{d-1})$ for surface integrals. Under the action of h-refinement with q fixed, the schemes yield an order of accuracy approximately $2q$. Under the action of q-refinement with the geometry fixed, the convergence is approximately exponential in q, i.e., doubling q approximately doubles the number of accurate digits.
- Complex domain topology is automatically detected by the algorithm, including, e.g., multi-component domains, tunnels, and intersections arising from multiple polynomial level sets, as well as corners, cusps, and other kinds of singularities. Instead of relying on an ad hoc subdivision/refinement process, the methods divide the integral domain into a finite number of (implicitly defined) pieces whose boundaries precisely align with the interfacial geometry.
- The overall approach is dimension independent; in particular, the same approach can be used in two- and three-dimensional applications.

A wide variety of accompanying numerical experiments assess the effectiveness of the quadrature algorithm. Examples include convergence studies under h-refinement and q-refinement; performance on randomly generated polynomials having non-trivial interfacial topology; ability to handle singularities such as cusps and self-intersections; adaptation to simplices instead of hyperrectangles; and junctions arising from overlapping implicitly defined domains.

An outline is as follows. In the next section, we briefly review some typical strategies for computing integrals of implicitly defined domains, and then provide some context motivating the design of the algorithms in this work. §3 presents the main algorithm, starting with an overview and then following with a discussion of individual driving components. A broad range of numerical tests is presented in §4, and concluding remarks with a brief discussion are given in §5. Meanwhile, the supplementary material accompanying this paper (available online) includes additional numerical experiments and algorithms.

2. Background

A wide variety of methods have been developed to compute integrals on implicitly defined domains. General approaches typically involve one of the following strategies; in each case, we indicate some of the associated challenges.

- *Methods which reconstruct an approximation of the interface.* For example, one approach is to build a piecewise linear approximation of the interface and then apply standard quadrature algorithms on the resulting simplices or polyhedra, see, e.g., [35,27,68]. Applied to complex interface geometry, this approach can have limited effectiveness due to a relatively low approximation power; often, subdivision techniques are required to locally refine the geometry approximation, and

often without a good return on the increased computational cost. Higher-order explicit reconstruction schemes are possible, see, e.g., [43,17,11,21,20,7]; while effective in some cases, the implementation of these methods quickly becomes more subtle as the approximation order is increased, in part due to a highly sensitive relationship between interface topology and Jacobian calculations.

- *Perturbation and correction methods.* These methods begin with a primary interface approximation, typically piecewise linear, and then incorporate a correction term representing the effect of locally projecting the approximation to the true curved interface geometry, see, e.g., [33,53,54,62,46]. In some cases, these methods offer a good compromise between the ease of piecewise linear reconstruction and the accuracy of a mildly higher-order interface representation. However, due to the reliance on the initial surface approximation, this kind of approach can breakdown on high curvature geometry or multiple surfaces in the same grid cell.

- *Application of smoothed Dirac delta and Heaviside functions.* These methods avoid an explicit reconstruction of the interface; instead, a sampling of the given level set function, typically on Cartesian grids, is used to approximate, e.g., $\int_\Gamma f = h^d \sum_i f(x_i) \delta_h(\phi(x_i)) |\nabla_h \phi(x_i)|$, where δ_h is a grid-dependent smoothed Dirac delta function and x_i are the grid points. These approaches typically rely on a cancellation of errors in the summation over regularly spaced grid points and it can be subtle to develop convergent schemes [63,16,58,64,77,36], especially higher-order ones [69–71]; see also related methods built through application of the co-area formula [15,31,76,66].

- *Mechanisms making use of integration by parts.* Quite distinct to the above approaches, another possibility is to use integration by parts (e.g., via the divergence theorem or the generalized Stokes theorem) to recast an implicitly defined volume/surface integral on the interior of a cell U as a surface integral on the boundary of U. In general, the domain of the resulting surface integral is itself implicitly defined, and so this process may continue recursively, removing one dimension at a time. Under certain assumptions, these methods can yield accurate and effective results, see, e.g., moment-based methods [38,55], their application to planar regions with piecewise parameterised boundaries [59,24], and trimmed parametric surfaces [23]. However, as discussed in [51], methods based on integration by parts may not always work, even for highly-resolved geometry—there is an implicit assumption that the boundary integrals on ∂U can detect enough of the internal geometry; in some cases, these methods require subdivision to prevent parts of the geometry from being missed.

- *Recasting as the graph of a height function.* Another possibility is to recast the implicitly defined geometry as the graph of an implicitly defined height function; using this interpretation, a quadrature scheme can then be formulated through the corresponding splitting of coordinates, i.e., through two nested integrals, one in the vertical/height direction, and the other in all remaining tangential directions. In the author's prior work [51,50], this approach was used to build very high-order quadrature schemes, e.g., 20th order; see also its extension to simplices by Cui et al. [13]. These methods can be fast, requiring only one-dimensional root finding, and inherit the high-order accuracy of the underlying one-dimensional Gaussian quadrature schemes. However, the overall approach relies on finding a suitable coordinate axis such that the interface is the graph of a well-defined, sufficiently smooth height function, e.g., without any corners, cusps, or vertical tangents. In [51,50], interval arithmetic is used to subdivide the hyperrectangle U, if necessary, until the level set function is provably monotone in a particular height direction k, thereby guaranteeing (by the implicit function theorem) a well-defined height function. This process is effective for a wide kind of geometry, but can fail in some cases, e.g., in the presence of singularities, corners, or junctions. In these situations, an excessive amount of subdivision can result, a process that may also need to be halted, potentially leaving behind small voids.

Putting aside the challenging cases of complex interface geometry, the above approaches can typically be applied to a broad class of sufficiently smooth level set functions. In the more restrictive case of polynomial level set functions, there is more structure that can be exploited, e.g., to effectively treat singularities. However, there appears to be relatively few works in the literature devoted specifically to building quadrature schemes for domains implicitly defined by polynomials. In some cases, these efforts have concentrated solely on computing the volume of semi-algebraic sets. For example, in Lairez et al. [32], and with a focus on arbitrary-precision calculations in algebraic geometry, the authors develop a method to compute volumes of compact semi-algebraic sets; the methods apply a dimension-reduction process to reformulate the volume computation as a solution of a high-order Picard-Fuchs differential equation, which in turn relies on high-precision ODE solvers. As another example, Henrion et al. [26] show how the volumes and moments of basic semi-algebraic sets can be approximated using a hierarchy of semi-definite programming problems, with guaranteed lower and upper bounds on these estimates, though the accuracy of these methods appear limited. A third example is of Hrivnák et al. [28], which build high-accuracy quadrature schemes using very few points, but only for integrals involving a specific kernel and for specific domains.

Our motivation in this work is to address some of the above-mentioned challenges, particularly in regard to handling complex interface geometry involving high degrees of curvature, non-trivial topology with multiple components, and characteristics such as corners, intersections, and cusps. To do so, we focus on implicitly defined domains given by multivariate polynomials, and then exploit the additional structure this representation affords. In particular, the quadrature algorithms developed in this work have the following design objectives in mind:

(i) Conceptually independent of the number of dimensions; e.g., the algorithms can be used for both two- and three-dimensional problems.

(ii) As close as possible to a "black box" algorithm; e.g., for use with dynamically-evolving interfacial geometry.

(iii) Avoid the use of subdivision; in particular, subdivision is permitted only when it can be proven that a finite (usually small) number of subregions is needed.

(iv) Effectively handle cases of junctions, corners, cusps, etc., ideally with near-exponential convergence in the number of quadrature points.

To achieve these objectives, we have adopted here a dimension reduction process in which the implicitly defined geometry is reinterpreted as the graph of an implicitly defined, potentially multi-valued height function. This approach is similar to the one used in the author's prior work [51], however the methods therein required a single-valued, well-defined and smooth height function. The objectives above, in particular (iii) and (iv), preclude this possibility and instead necessitate multi-valued height functions having additional kinds of characteristics, which in turn require additional mechanisms to effectively handle, as discussed next.

3. Quadrature on domains implicitly defined by multivariate polynomials

3.1. Main formulation

As motivated in the prior section, our essential approach is to apply a dimension reduction technique wherein the implicitly defined interface Γ is recast as the graph of an implicitly defined multi-valued "height" function. Once a suitable coordinate axis representing the vertical height direction is chosen, a quadrature scheme is built by performing: (a) an integration in the height direction along with (b) an integration in the tangential direction(s). Part (a) applies one-dimensional quadrature schemes to compute vertical line integrals, subdivided into pieces according to the location of the interface, while part (b) is itself recast as an integral over implicitly defined regions, in one fewer dimensions, whose associated integrand evaluates part (a). The latter integral is called the "base integral" and its domain shall be implicitly defined by a collection of multivariate polynomials; as such, the quadrature algorithm can execute itself and proceed recursively, reducing the number of dimensions one at a time. In order to obtain high quadrature accuracy, it is important to appropriately identify and separate the different regions of behaviour of the multi-valued height function. Changes in behaviour typically correspond to wherever the interface crosses the upper or lower faces of the constraint domain U, wherever two surfaces cross each other, or at branch points for which two surfaces join together at a vertical tangent. These aspects will be discussed shortly, but first we establish some preliminaries.

Throughout this section we take, for ease of exposition, a reference domain equal to the d-dimensional unit cube; adaptation to any hyperrectangular domain $U \subset \mathbb{R}^d$ is a straightforward modification. For the time being, we will also focus attention on the case of volumetric integrals; surface integrals will be treated at the end of the section in §3.7. As such, suppose we are given one or more polynomial functions $\phi_i : (0,1)^d \to \mathbb{R}$, $i = 1, \dots, n$, which together implicitly define an interface $\Gamma := \bigcup_{i=1}^n \{\phi_i = 0\}$. Suppose also that we are given an integrand $f : (0,1)^d \to \mathbb{R}$, which is permitted to exhibit discontinuities across the interface, but is otherwise assumed to be sufficiently smooth. For the moment, let us also suppose we have identified a suitable coordinate axis representing the height direction, e_k, say.[1] Our task is to build a quadrature scheme to integrate f over the volumetric region on either side of Γ, constructed by performing an outer integral (over $(0,1)^{d-1}$) of an inner 1D integral in the vertical direction e_k.

Denote by $x = (x_1, \dots, x_{k-1}, x_{k+1}, \dots, x_d) \in (0,1)^{d-1}$ a point in the $(d-1)$–dimensional cube, and denote by $x + ye_k$ the point $(x_1, \dots, x_{k-1}, y, x_{k+1}, \dots, x_d)$, i.e., the lifting of x in the direction e_k by the amount y. We denote the integrand of the base integral by $\mathcal{F} : (0,1)^{d-1} \to \mathbb{R}$. In particular, given a base point x, evaluation of $\mathcal{F}$ at x performs an integral over a vertical line segment $\{x + ye_k : 0 \le y \le 1\}$; since our goal is to build high order quadrature schemes which can separately handle either side of Γ, it follows that the inner integral should be divided into pieces, demarcated by interfacial crossing points. This can succinctly be described by asking the inner integral to build a quadrature scheme which integrates f on the connected components of the set $(0,1) \setminus \Gamma(x)$, where $\Gamma(x) := \bigcup_{i=1}^n \{y : 0 \le y \le 1 \text{ and } \phi_i(x + ye_k) = 0\}$. Here, it is convenient to introduce a custom notation: denote by an integral symbol with overset $\diamond$ as summing over the connected components of the corresponding domain:
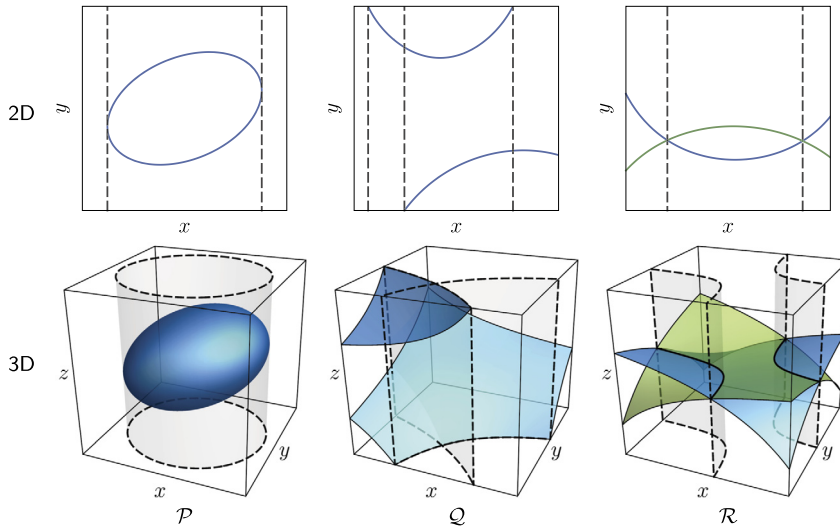
$$\oint_\Omega := \sum_j \int_{\Omega_j} \quad \text{where } \Omega_j \text{ enumerates the connected components of } \Omega. \tag{2}$$

Using this notation, the base integral's integrand $\mathcal{F} : (0,1)^{d-1} \to \mathbb{R}$ is given by

$$\mathcal{F}(x) := \oint_{(0,1) \setminus \Gamma(x)} f(x + ye_k)\, dy \tag{3}$$

and part of the algorithm's role is to produce a quadrature scheme which approximately evaluates $\mathcal{F}(x)$ for a given x.

---

[1] e_i denotes the standard basis vector in the direction of the ith coordinate.

Fig. 1. Depictions of the changes in behaviour of Γ according to its representation as the graph of a multi-valued height function. In each case, the height direction is "up" (i.e., $\hat{y}$ in 2D and $\hat{z}$ in 3D), the blue and green lines/surfaces illustrate the interface, and the dashed lines/surfaces represent the vertical extrusion of the sets $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{R}$, from the base cube $(0,1)^{d-1}$ into the whole cube $(0,1)^d$. *(left column)* $\mathcal{P}$ denotes the set of base points above which the interface branches, typically occurring at the vertical tangents of rounded surfaces. *(middle column)* $\mathcal{Q}$ denotes the set of base points above which the interface crosses the upper or lower faces of $(0,1)^d$, i.e., wherever the multi-valued height function has one or more values equal to 0 or 1. *(right column)* $\mathcal{R}$ denotes the set of base points above which two different interfaces cross. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

Note that the smoothness of $\mathcal{F}(x)$ depends on changes in the connected component behaviour of $\Gamma(x)$, which are precisely the changes in the behaviour of Γ according to its representation as the graph of a multi-valued height function. To illustrate, Fig. 1 demonstrates some typical scenarios in 2D and 3D. In each case, the coordinate axis representing the vertical height direction is "up", i.e., $\hat{y}$ in 2D and $\hat{z}$ in 3D, and the points of interest are subsets of the base cube $(0,1)^{d-1}$. Points labelled $\mathcal{Q}$ indicate where the interface Γ crosses the upper or lower faces, and correspond to where one or more values of the height function equals 0 or 1. Points labelled $\mathcal{R}$ indicate the base points above which two different interfaces cross. Points labelled $\mathcal{P}$ indicate base points above which the height function branches; typically, branching occurs at vertical tangents of rounded surfaces such as circles and spheres.

Combined, these points mark the boundaries of the piecewise-smooth behaviour of $\mathcal{F}(x)$. To attain high-order accuracy in the overall quadrature scheme, it follows that the base integral should be prevented from "jumping across" these boundaries. We achieve this through a division of the base integral's domain into one or more pieces, such that the base integral produces an accurate quadrature scheme for each piece. As before, this can be succinctly formulated through the use of connected components of an implicitly defined domain; specifically, we begin with the base cube $(0,1)^{d-1}$ and then subtract out an implicit description of the points $\mathcal{P} \cup \mathcal{Q} \cup \mathcal{R}$ discussed above; in essence, the points $\mathcal{P} \cup \mathcal{Q} \cup \mathcal{R}$ form the implicitly defined interface of the $(d-1)$-dimensional base integral.

Implicitly characterising points in $\mathcal{Q}$, i.e., changes in the smoothness of $\mathcal{F}$ arising from interfacial crossings at the upper and lower faces, is straightforward: this can be accomplished by excluding the sets $\{x : \phi_i(x+0e_k) = 0\}$ and $\{x : \phi_i(x+1e_k) = 0\}$ from the base integral's domain.

An implicit characterisation of $\mathcal{P}$ and $\mathcal{R}$ requires additional tools, namely *discriminant* and *resultant* methods. A key tool in algebraic geometry, these methods take in as input one or two polynomials and output a new polynomial whose roots pinpoint the location of the shared roots of the input (and its derivative, in the case of discriminants). More details on their use will be given in §3.4; for now, it suffices to state their application yields multivariate polynomials, in one fewer dimensions, whose zero level sets contain the set of points $\mathcal{P}$ and $\mathcal{R}$. These polynomials, in conjunction with the upper- and lower-face restrictions of ϕ_i used to characterise $\mathcal{Q}$, complete the partitioning of the base integral's domain into one or more connected components.

The output of the above analysis is a set of polynomials $\psi_j : (0,1)^{d-1} \to \mathbb{R}$ describing the domain of the base integral in the form $(0,1)^{d-1} \setminus \bigcup_j \{\psi_j = 0\}$. Using the notation of (2), the base integral then takes the form

$$\oint_{(0,1)^{d-1} \setminus \bigcup_j \{\psi_j = 0\}} \mathcal{F}(x)\,dx.$$

Note that this is of the same form as we started with, but in one fewer dimensions: the task of the base integral is to build a quadrature scheme for an implicitly defined domain, whose interface is the union of a collection of polynomial isosurfaces, and whose integrand is, by construction, smooth on either side of that interface. Therefore, we can apply the

quadrature algorithm recursively, eliminating one coordinate direction at a time, until we reach the conceptually simple one-dimensional base case.

We are nearly in a position to give a high-level description of the quadrature algorithm, however some aspects need additional motivation:

- *Quadrature methods.* By construction, we need only apply one-dimensional quadrature schemes, but there is a choice in the specific kind of scheme to employ. Indeed, it can be worthwhile to apply different kinds of 1D quadrature in different parts of the dimension-reduction process. In this work we have mainly explored the use of Gauss-Legendre and tanh-sinh quadrature, choosing between the two depending on the smoothness of the height functions involved. This choice is one of the main topics of discussion in this paper, and some guidelines will be given.
- *Polynomial basis.* Numerically, a basis for multivariate polynomials must be chosen. Given the dimension-reduction process and our choice of working with hyperrectangular domains, a natural candidate is a tensor-product basis of polynomials. In particular, we have made extensive use of the Bernstein basis, owing to its favourable numerical properties such as well-conditioned root finding and accurate range/bounds evaluation; more comments are given in §3.2.
- *Polynomial elimination and masking.* It can be highly beneficial to detect when polynomials can be eliminated in the dimension reduction process. For example, if we can determine that Γ is empty, then a simple tensor-product quadrature scheme would suffice. As another example, if we can detect that no branching points exist (e.g., points labelled $\mathcal{P}$ in Fig. 1), then two benefits arise: (i) we can avoid calculating discriminants, and (ii) conclude that the associated height function has no vertical tangents, thereby helping to decide which quadrature scheme to use. In this work, these kinds of checks are performed with efficient "mask" operations which, in turn, exploit useful properties of the Bernstein basis. Roughly speaking, a mask divides the cube $(0, 1)^d$ into a regular grid of $\mathcal{M} \times \cdots \times \mathcal{M}$ of subcells (typically $\mathcal{M} = 4$ or 8); on each subcell, the mask has a binary value of 0 and 1 and indicates whether its accompanying polynomial is provably nonzero on the subcell or if its roots can be ignored. More details on masks and their use are given in §3.3.
- *Choice of height direction/elimination axis.* Prior to applying the dimension-reduction process, one must make an appropriate choice for the height direction, i.e., the coordinate axis to be eliminated, and in most applications, this choice will need to be made algorithmically. A variety of aspects could impact this decision, including: which directions yield height functions with the smallest amount of curvature (and thus highest quadrature accuracy); which directions yield branching points/vertical tangents (which should generally be avoided, if possible); and which directions lead to the fewest number of polynomials in the description of the base integral's implicitly defined domain (which can benefit efficiency). Some of these aspects can be assessed with the masking operations. Further remarks are given in §3.5.

With these preliminaries, a high-level description of the overall quadrature scheme is given in Algorithm 1 and Algorithm 2. Various core operations in these algorithms have yet to be discussed in detail; we shall do so in the forthcoming sections. Here, a few remarks are in order:

- Algorithm 1 is the main engine and follows the dimension-reduction approach motivated above: it chooses a height direction/elimination axis k, formulates an implicitly-defined domain for the base integral, and creates a new integrand $\mathcal{F}$ for the base integral. Meanwhile, the role of Algorithm 2 is to evaluate the constructed integrands.
- The dimension reduction process is started by executing Algorithm 1 on the user-provided multivariate polynomials (often just one of them). As coordinate axes are eliminated one by one, eventually the base case in $d = 1$ dimensions is reached; the base case executes Algorithm 2 which builds a 1D quadrature scheme and evaluates the associated integrand at these points; in turn, a new 1D quadrature scheme is executed, and this process continues recursively, adding one dimension at a time. By construction, the result is a tensor-product quadrature scheme warping to the geometry found along the way. Although these algorithms assumed that a top-level integrand $f : (0, 1)^d \to \mathbb{R}$ was provided by the user, they can easily be modified to record the final quadrature scheme as a collection of points $x_i \in (0, 1)^d$ with associated weights $w_i \in \mathbb{R}^+$.
- From an object-oriented point of view, one may interpret the first algorithm as constructing integrand function objects, i.e., functors, whose evaluation is defined by the second algorithm. In fact, many of the operations in Algorithm 1 are independent of the evaluation of the quadrature scheme; as such, one can cache the results and execute with different quadrature orders q, different 1D quadrature schemes, etc.

Algorithms 1 and 2 represent the main driver routines for the quadrature algorithms developed in this work. In the remainder of this section, we discuss in more detail some of the tools and algorithms needed to fully realise their implementation. A summary of the development is then given at the conclusion of this section, prior to the presentation of an extensive range of test cases in §4.

3.2. Bernstein basis

Numerically, a basis must be chosen for the multivariate polynomials being manipulated by Algorithms 1 and 2. We have adopted here a Bernstein basis, owing to its many favourable properties:

Algorithm 1 Compute $\mathcal{I}(f; \{(\phi_i, m_i)\}_{i=1}^n, q)$. Given a list of multivariate polynomials $\phi_i : (0, 1)^d \to \mathbb{R}$ and associated masks $m_i : \mathbb{N}_{\mathcal{M}}^d \to \{0, 1\}$, a function $f : (0, 1)^d \to \mathbb{R}$, and the order of quadrature q, approximate the volume integral $\oint_{(0,1)^d \backslash \Gamma} f(x) \, dx$, where $\Gamma := \bigcup_i \{\phi_i = 0\}$.

1: **if** $d = 1$ **then**
2:      Execute one-dimensional base case using the empty set as input base point:
3:      **return** $\mathcal{F}_{\text{eval}}(\varnothing; 1, f, \{(\phi_i, m_i)\}_{i=1}^n, q)$.
4: Choose a height direction/elimination axis $k \in \{1, \dots, d\}$ (see §3.5).
5: Define Ψ to be a list of polynomial and mask pairs, initially empty.
   Examine the lower and upper face restrictions of input polynomials:
6: **for** each i **do**
7:      **for** $y \in \{0, 1\}$ **do**
8:          Define $\psi : (0, 1)^{d-1} \to \mathbb{R}$ by the face restriction of ϕ_i, i.e., $\psi(x) = \phi_i(x + y e_k)$.
9:          Compute $m := \texttt{nonzeroMask}(\psi, \texttt{faceRestriction}(m_i, k, y))$.
10:          **if** m is nonempty **then**
11:              Add (ψ, m) to the list Ψ.
   Examine the pseudo-discriminants of input polynomials:
12: **for** each i **do**
13:      Compute the polynomial $\partial_k \phi_i$.
14:      Compute $m := \texttt{intersectionMask}((\phi_i, m_i), (\partial_k \phi_i, m_i))$.
15:      **if** m is nonempty **then**
16:          Compute the pseudo-discriminant $\psi := \Delta(\phi_i; k)$.
17:          Add $(\psi, \texttt{collapseMask}(m, k))$ to the list Ψ.
   Examine the pairwise resultants of input polynomials:
18: **for** each pair i and j with $i \neq j$, ignoring order **do**
19:      Compute $m := \texttt{intersectionMask}((\phi_i, m_i), (\phi_j, m_j))$.
20:      **if** m is nonempty **then**
21:          Compute the resultant $\psi := R(\phi_i, \phi_j; k)$.
22:          Add $(\psi, \texttt{collapseMask}(m, k))$ to the list Ψ.
   Apply dimension reduction:
23: Define a new integrand $\mathcal{F} : (0, 1)^{d-1} \to \mathbb{R}$ by $\mathcal{F}(x) := \mathcal{F}_{\text{eval}}(x; k, f, \{(\phi_i, m_i)\}_{i=1}^n, q)$.
24: **return** $\mathcal{I}(\mathcal{F}; \Psi, q)$.

Algorithm 2 Compute $\mathcal{F}_{\text{eval}}(x; k, f, \{(\phi_i, m_i)\}_{i=1}^n, q)$. Given a point $x \in \mathbb{R}^{d-1}$, a height direction k, a function $f : \mathbb{R}^d \to \mathbb{R}$, a list of multivariate polynomials $\phi_i : (0, 1)^d \to \mathbb{R}$ and associated masks $m_i : \mathbb{N}_{\mathcal{M}}^d \to \{0, 1\}$, and the order of quadrature q, approximate the one-dimensional integral $\oint_{(0,1)\backslash\Gamma(x)} f(x + y e_k) \, dy$, where $\Gamma(x) := \bigcup_i \{y : 0 \leq y \leq 1 \text{ and } \phi_i(x + y e_k) = 0\}$.

1: Define $\mathcal{R}$ to be a list, initialised by $\mathcal{R} := \{0, 1\}$.
2: **for** each i **do**
3:      Define $\psi : (0, 1) \to \mathbb{R}$ by the vertical line restriction of ϕ_i above x in the direction k, i.e., $\psi(y) := \phi_i(x + y e_k)$.
4:      Compute the set of real roots of ψ in the interval $(0, 1)$.
5:      **for** each such root y **do**
6:          Let $j \in \mathbb{N}_{\mathcal{M}}^d$ denote the mask subcell containing the point $x + y e_k$.
7:          **if** $m_i(j) = 1$ **then**
8:              Add y to the list $\mathcal{R}$.
9: Sort $\mathcal{R} = \{r_1, \dots, r_\ell\}$ into ascending order such that $0 = r_1 < r_2 \leq \cdots \leq r_{\ell-1} < r_\ell = 1$.
10: Set $I = 0$.
11: **for** $j = 1$ to $\ell - 1$ **do**
12:      **if** $[r_j, r_{j+1}]$ is non-degenerate **then**
13:          On this interval, apply the chosen 1D quadrature scheme of order q (e.g., Gauss-Legendre or tanh-sinh); let $x_{q,i}$ and $w_{q,i}$ denote the quadrature nodes and weights, respectively, relative to the interval (r_j, r_{j+1}):
14:          Update $I := I + \sum_{i=1}^q w_{q,i} f(x + x_{q,i} e_k)$.
15: **return** I.

- Computing roots of polynomials in the Bernstein basis is generally well-conditioned. In fact, for simple roots in $[0, 1]$ of an arbitrary polynomial, the root condition number is smaller in the Bernstein basis than in any other non-negative polynomial basis, including the power/monomial basis [19,18].
- One of the most accurate (and straightforward) methods for range evaluation of a polynomial is through the Bernstein basis [22,45,22,34,5]. These range evaluation methods employ the convex hull property of Bernstein polynomials and guarantee that a polynomial's value, at any point its the rectangular reference domain, is no smaller (or larger) than its minimum (or maximum) coefficient; in fact, these bounds become monotonically more accurate under the (stable) operations of Bernstein degree elevation and subdivision [22,34].
- Simple and stable algorithms exist to manipulate polynomials in Bernstein form, including evaluation, differentiation, degree elevation, and subdivision, among a number of other useful operations; see, e.g., [18].

For the dimension reduction algorithm considered in this work, it is natural to employ a tensor-product Bernstein basis. In d dimensions, let $(n_1, \dots, n_d)$ denote the degree of a Bernstein polynomial ϕ; then ϕ takes the form

$$\phi(x_1, \ldots, x_d) = \sum_{i_1=0}^{n_1} \cdots \sum_{i_d=0}^{n_d} c_{i_1,\ldots,i_d} b_{i_1}^{n_1}(x_1) \cdots b_{i_d}^{n_d}(x_d) \tag{4}$$

where $b_\ell^n(x) = \binom{n}{\ell}(1-x)^{n-\ell}x^\ell$, $\ell = 0, \ldots, n$, are the Bernstein basis functions of degree n and $c_i = c_{i_1,\ldots,i_d}$ are the Bernstein coefficients of ϕ. This formulation represents ϕ in coordinates relative to the standard unit domain $[0, 1]^d$; subdivision refers to transforming the coefficients relative to a subset hyperrectangle, and can be stably computed using the *de Casteljau* algorithm. This operation, along with many other standard operations in Bernstein form (such as differentiation, degree elevation, multiplication, etc.) are usually given for one-dimensional Bernstein polynomials, see, e.g., [18], but it is straightforward to extend them to multivariate tensor-product polynomials of mixed degrees.

3.3. Masking

A number of operations in Algorithm 1 benefit from a cheap and robust method to isolate zeros of a polynomial or to detect when two polynomials have non-intersecting zero isosurfaces; for example, if it can be shown that all of the given polynomials ϕ_i are nonzero, then $\Gamma = \bigcup_i \{\phi_i = 0\}$ is empty and a simple tensor-product quadrature scheme for $(0, 1)^d$ would suffice. However, our main use for these root isolation methods are to: (i) detect when Γ is the graph of a multi-valued height function devoid of branching (to inform the type of 1D quadrature to use); (ii) avoid the calculation of a resultant or discriminant (which is potentially expensive in comparison to other aspects of the algorithm); and (iii) mitigate unnecessary subdivision caused by unwanted roots of resultant and discriminant calculations. These aspects will motivated in subsequent sections.

A *mask* $m : \mathbb{N}_{\mathcal{M}}^d \to \{0, 1\}$ is a binary-valued function defined on a $\mathcal{M} \times \cdots \times \mathcal{M}$ grid of subcells Ω_i, where the ith subcell of $[0, 1]^d$ is given by $\Omega_i = \prod_{\ell=1}^d [\frac{1}{\mathcal{M}}(i_\ell - 1), \frac{1}{\mathcal{M}}i_\ell]$.[2] Roughly speaking, a mask is built from one or two polynomials and helps to isolate the interfacial geometry of those polynomial(s)—a subcell value of "0" means the subcell can be (provably) ignored, while a value of "1" indicates that the subcell likely contains interfacial geometry. More concretely, the binary values of a mask m are related to its associated polynomial ϕ in the following way: $m(i) = 0$ if it can be proven that ϕ is nonzero on $\Omega_i \pm \epsilon$; $m(i) = 0$ if it can be proven that any zeros of ϕ in Ω_i can be ignored by vertical line integrals (see §3.4); and $m(i) = 1$ otherwise. Here, $\Omega_j \pm \epsilon$ denotes a slightly expanded version of Ω_j, overlapping with neighbouring subcells by a small amount ϵ: this overlap is used to eliminate edge/border cases that could be affected by roundoff error in floating point arithmetic.

These masks are (cheaply) computed using convex hull properties of Bernstein polynomials and relate to their effectiveness in accurately computing polynomial ranges/bounds cited earlier in §3.2. The design of these algorithms is deferred to the supplementary material; here, we summarise the essential ideas pertinent to their use in Algorithms 1 and 2 and the following sections:

- Each polynomial in the dimension reduction process, whether it be upper- or lower-face restrictions, discriminants, or resultants, has a mask associated with it. Typically, the quadrature algorithm is executed with the user-provided driving polynomial(s) (in full dimensional space) with masks identically equal to 1 (as an easy means to convey that any subcell may contain the interface). In Algorithm 1, a mask m is deemed "nonempty" if and only if $m(i) = 1$ on at least one subcell i.
- Given a polynomial f and mask f_m, the routine `nonzeroMask`(f, f_m) uses "orthant tests" to return a new mask m such that $m(i) = 0$ if and only if at least one of the following conditions hold: (i) $f_m(i) = 0$; or (ii) it can be proven that f does not have any real roots in the (slightly expanded) subcell $\Omega_i \pm \epsilon$. This routine can be used to determine which subcells contain the interface of f; in particular, if the resulting mask is empty, i.e., identically zero, then the polynomial f can be removed from the dimension reduction process.
- Given two polynomials f and g, with masks f_m and g_m, the routine `intersectionMask`$\big((f, f_m), (g, g_m)\big)$ uses orthant tests to return a new mask m such that $m(i) = 0$ if and only if at least one of the following three conditions hold: (i) $f_m(i) = 0$; or (ii) $g_m(i) = 0$; or (iii) it can be proven that f and g do not share any real roots in the (slightly expanded) subcell $\Omega_i \pm \epsilon$. This routine is used to quickly ascertain whether two polynomials have nonintersecting zero level sets as well as to help eliminate unwanted zeros of resultant/discriminant calculations, as explained in §3.4.
- The routine `collapseMask`(m, k) takes a d-dimensional mask and collapses it to a $(d-1)$-dimensional mask by bitwise-OR-ing along each column in the direction k, i.e., it returns $m_c : \mathbb{N}_{\mathcal{M}}^{d-1} \to \{0, 1\}$ such that $m_c(i) = 0$ if and only if $m(i + \ell e_k) = 0$ for every $\ell = 1, \ldots, \mathcal{M}$. This routine, along with the related method `faceRestriction` (below), is a kind of dimension-reduction analogue for masks: in Algorithm 1, these routines create a mask for the polynomials making up the domain of the base integral.
- Finally, the routine `faceRestriction`(m, k, y) takes a d-dimensional mask and returns its restriction to the lower $(y = 0)$ or upper $(y = 1)$ face in the direction k; in other words, the routine returns the mask $m_{\text{face}} : \mathbb{N}_{\mathcal{M}}^{d-1} \to \{0, 1\}$ where $m_{\text{face}}(i) = m\big(i + [1 + y(\mathcal{M} - 1)]e_k\big)$.

---

[2] $\mathbb{N}_{\mathcal{M}} := \{1, \ldots, \mathcal{M}\}$ denotes the first $\mathcal{M}$ positive integers.

The number of subcells used to compute these masks, $\mathcal{M}$, is a user-chosen parameter. With increasing $\mathcal{M}$, the accuracy of the associated Bernstein polynomial range evaluation increases, and thus so too the degree of confidence in isolating zeros of the involved polynomials. However, our experiments indicate that there is little benefit in employing a large $\mathcal{M}$, as a small $\mathcal{M}$ is often sufficiently effective in achieving the objectives of the masking operations. In particular, a value of 4 or 8 works well, and $\mathcal{M} = 8$ has been used throughout this work.[3] It is important to stress, however, that the "correctness" of the quadrature algorithms is independent of $\mathcal{M}$, meaning that the correct topology of the interface will always be determined by Algorithms 1 and 2, even if no masks are used. The main benefit of the masks is to eliminate unwanted zeros of resultant/discriminant calculations, which results in less unnecessary interval subdivisions in Algorithm 2. Some of these aspects are discussed next.

3.4. Resultants and pseudo-discriminants

Recall that the domain of the base integral is divided into pieces corresponding to the connected components of $(0, 1)^{d-1} \setminus (\mathcal{P} \cup \mathcal{Q} \cup \mathcal{R})$, where (see Fig. 1): $\mathcal{Q}$ indicate the base points of upper- or lower-face crossings of the interface; $\mathcal{R}$ indicate the base points above which two interfaces cross; and $\mathcal{P}$ indicate base points above which the height function branches, typically at the vertical tangents of rounded surfaces. In all three cases, these sets can be implicitly defined by multivariate polynomials, with $\mathcal{Q}$ being a simple case of restricting the input polynomial(s) to the upper- and lower-faces. In the cases of $\mathcal{P}$ and $\mathcal{R}$, these sets can be implicitly defined through the use of polynomial resultant methods, as follows.

3.4.1. Resultants of univariate Bernstein polynomials
Given two univariate polynomials f and g, a *resultant* $R(f, g)$ is a polynomial expression in the coefficients of f and g such that $R(f, g) = 0$ if and only if f and g share a common root (whether real or complex). A number of resultant formulations are possible; in many cases they are given by determinants of a Sylvester or Bézout matrix. These matrices are often stated in terms of the power/monomial basis coefficients of f and g, however a number of recent works [72,73,6,74] have derived formulations for use with the Bernstein basis:

- *Sylvester resultant for Bernstein polynomials.* Here, we adopt the formulation given in [74]: the Sylvester resultant matrix, of size $(n + m) \times (n + m)$, corresponding to degree n and m polynomials $f(x) = \sum_{i=0}^{n} f_i b_i^n(x)$ and $g(x) = \sum_{i=0}^{m} g_i b_i^m(x)$, is given by

$$
S(f, g) = \begin{bmatrix}
f_0 \binom{n}{0} & f_1 \binom{n}{1} & f_2 \binom{n}{2} & \cdots & & \\
& f_0 \binom{n}{0} & f_1 \binom{n}{1} & f_2 \binom{n}{2} & \cdots & \\
& & f_0 \binom{n}{0} & f_1 \binom{n}{1} & \ddots & \\
& & & \ddots & \ddots & \\
g_0 \binom{m}{0} & g_1 \binom{m}{1} & g_2 \binom{m}{2} & \cdots & & \\
& g_0 \binom{m}{0} & g_1 \binom{m}{1} & g_2 \binom{m}{2} & \cdots & \\
& & g_0 \binom{m}{0} & g_1 \binom{m}{1} & \ddots & \\
& & & \ddots & \ddots &
\end{bmatrix} D^{-1},
$$

  where $D = \mathrm{diag}\big(\{\binom{m+n-1}{i}\}_{i=0}^{m+n-1}\big)$ is a diagonal scaling.[4] Each row of $S(f, g)$ contains a shifted copy of the scaled Bernstein coefficients of f and g, with the first m rows corresponding to f and the next n rows corresponding to g. Using the Sylvester matrix, a resultant expression is then given by

  $$R(f, g) = \det S(f, g).$$

- *Bézout resultant matrix for Bernstein polynomials.* The (symmetric) Bézout resultant matrix $B(f, g) = (b_{ij})_{i,j=1}^{n}$, of size $n \times n$, corresponding to degree n polynomials $f(x) = \sum_{i=0}^{n} f_i b_i^n(x)$ and $g(x) = \sum_{i=0}^{n} g_i b_i^n(x)$ can be defined recursively[5] [6] such that

---

[3] Profiling indicates that, for this choice of $\mathcal{M}$, mask computations represent a small to medium portion of the overall computational cost of the quadrature algorithm.

[4] For an intuitive derivation of the Sylvester matrix $S(f, g)$ for Bernstein polynomials, see [18]; this derivation does not include the diagonal scaling D, which can be effective in preventing an unnecessary rapid growth of the matrix entries for large degree polynomials, improving numerical conditioning.

[5] In floating point arithmetic, it is important to first build the lower left triangular portion of $B(f, g)$, and then copy the result (by symmetry) into the upper right triangular portion; a direct application of the recursive formula to compute the upper right triangular portion is numerically unstable and can result in a blowup of rounding errors.

$$\mathsf{b}_{i,1} = \tfrac{n}{i}(f_i g_0 - f_0 g_i), \qquad\qquad\qquad 1 \le i \le n,$$

$$\mathsf{b}_{n,j+1} = \tfrac{n}{n-j}(f_n g_j - f_j g_n), \qquad\qquad\quad 1 \le j \le n-1,$$

$$\mathsf{b}_{i,j+1} = \tfrac{n^2}{i(n-j)}(f_i g_j - f_j g_i) + \tfrac{j(n-i)}{i(n-j)}\mathsf{b}_{i+1,j}, \qquad 1 \le i, j \le n-1.$$

As before, a resultant expression is then given by

$$R(f, g) = \det B(f, g).$$

Comparing these two approaches, the Bézout method has a number of potentially useful properties including, e.g., it results in a smaller matrix, and thus a faster determinant calculation; it can also exhibit better numerical conditioning than the Sylvester method [75]. However, it requires equal degree polynomials, thus requiring degree elevation of one of the polynomials in the mixed degree case, whereas the Sylvester method is directly applicable to mixed degrees. In this work, we have used the Bézout method whenever the input polynomials have equal degree, and the Sylvester method otherwise.

3.4.2. Resultants of multivariate Bernstein polynomials

The above univariate resultant methods can be adapted to multivariate Bernstein polynomials, as follows. Let $f(x) = \sum_i f_i \prod_{\ell=1}^d b_{i_\ell}^{n_\ell}(x_\ell)$ and $g(x) = \sum_i g_i \prod_{\ell=1}^d b_{i_\ell}^{m_\ell}(x_\ell)$ denote two tensor-product Bernstein polynomials of degrees $(n_1, \dots, n_d)$ and $(m_1, \dots, m_d)$, respectively. Suppose that $k \in \{1, \dots, d\}$ is a chosen elimination axis. Then, f and g can be reinterpreted as univariate polynomials of degree n_k and m_k, with their coefficients a tensor-product Bernstein polynomial in one fewer dimensions. Using the lifting notation defined earlier, for a base point $\bar{x} = (x_1, \dots, x_{k-1}, x_{k+1}, \dots, x_d)$, denote these univariate polynomials by $f_{\bar{x}}$ and $g_{\bar{x}}$, such that $f_{\bar{x}}(y) = f(\bar{x} + y e_k)$ for all y, and similarly for $g_{\bar{x}}$. The above univariate resultant method can then be applied to $f_{\bar{x}}$ and $g_{\bar{x}}$ to obtain a resultant $R(f_{\bar{x}}, g_{\bar{x}})$. Importantly, the output of this operation is a polynomial in the coefficients of $f_{\bar{x}}$ and $g_{\bar{x}}$, and thus a polynomial in $\bar{x}$; denoting this polynomial by $R(f, g; k)$ with the argument k indicating which variable or axis has been eliminated, we have

$$\big[R(f, g; k)\big](\bar{x}) = R(f_{\bar{x}}, g_{\bar{x}}) \text{ for all } \bar{x}. \tag{5}$$

For an arbitrary pair of tensor-product polynomials f and g, the resultant $R(f, g; k)$ will also be a tensor-product polynomial. The maximum possible degree of this polynomial, $(r_1, \dots, r_{k-1}, r_{k+1}, \dots, r_d)$, is given by

$$r_\ell = n_\ell m_k + m_\ell n_k, \quad \text{for} \quad \ell = 1, \dots, k-1, k+1, \dots, d. \tag{6}$$

A number of different approaches could be used to compute $R(f, g; k)$. For example, one could apply a division-free determinant algorithm together with Bernstein multiplication, to compute the determinant of the Sylvester or Bézout matrix for all $\bar{x}$ at once. In this work, we have instead opted for a faster method (in terms of complexity): the method uses a Bernstein interpolation algorithm applied to a pointwise nodal evaluation of $R(f_{\bar{x}}, g_{\bar{x}})$ at sufficiently many points to recover a tensor-product Bernstein polynomial of degree given by (6). Details on this algorithm are given in the supplementary material.

3.4.3. Pseudo-discriminants

Similar to the resultant, another key tool in algebraic geometry is the *discriminant*. Given a polynomial f, the discriminant $\Delta(f)$ is a polynomial expression in the coefficients of f such that $\Delta(f) = 0$ if and only if f has a double root. One possible method for defining the discriminant is by a normalisation of the resultant of f and its derivative, with the specific kind of normalisation sometimes varying from author to author. In this work, however, we deliberately choose to leave the expression unnormalised—in particular, we define the *pseudo-discriminant* Δ of a multivariate polynomial f in the direction k as

$$\Delta(f; k) := R(f, \partial_k f; k).$$

To illustrate the usefulness of this choice, consider the 2D polynomial $\phi(x, y) = xy$, whose zero level set is a cross aligning with the Cartesian coordinate axes. For the purposes of quadrature, supposing the height direction is $\hat{y}$, say, it is important to detect the location of the vertical interface, so that an accurate quadrature scheme can be built for all four quadrants. To do so, we can use the pseudo-discriminant of ϕ in the direction $\hat{y}$, whose calculation yields the 1D polynomial $\Delta(\phi; y) = R(\phi, \partial_y \phi; y) = x$; note that the zero set of this polynomial successfully pinpoints the vertical interface of ϕ. On the other hand, the conventional discriminant of ϕ (using one of the most common kinds of normalisation) with elimination variable y yields $\Delta(\phi; y) = 1$; as such, it does not provide any useful information about the position of the vertical interface. In a sense, a vertical interface is a degenerate example of a branching point (labelled $\mathcal{P}$ in Fig. 1); the conventional discriminant may not always pick up these points, but the pseudo-discriminant can, hence its adoption in the quadrature algorithms of this work. To compute $\Delta(f; k)$, the same algorithm as for the resultant can be used; for an arbitrary tensor-product polynomial f of Bernstein degree $(n_1, \dots, n_d)$, the output will also be a tensor-product polynomial, and its maximum possible degree $(r_1, \dots, r_{k-1}, r_{k+1}, \dots, r_d)$ is given by

$$r_\ell = (2n_k - 1)n_\ell, \quad \text{for} \quad \ell = 1, \dots, k-1, k+1, \dots, d. \tag{7}$$
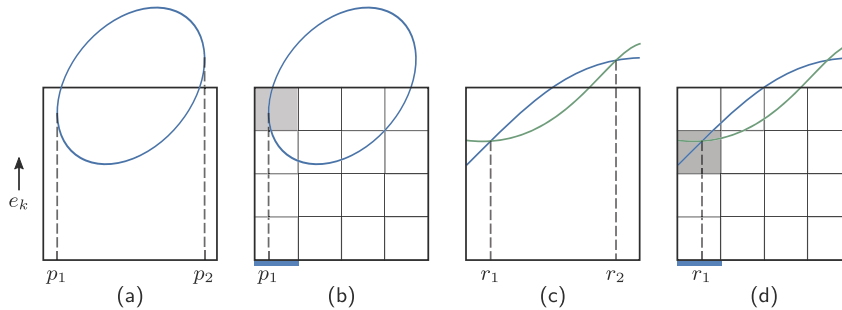
Fig. 2. Superfluous zeros of resultant calculations and their mitigation through masking methods. *(a)* The discriminant of a rotated ellipse yields a univariate polynomial with two real roots, p_1 and p_2; in particular, p_2 corresponds to a branching point of the interface located outside the domain of interest. *(b)* Using the masking methods of §3.3, here illustrated with $\mathcal{M} = 4$, the discriminant polynomial is associated with a mask which is enabled (value 1) in the shaded cell, and disabled (value 0) everywhere else. The restriction of this mask to the base domain yields a one-dimensional mask with one enabled cell (blue). Combined, the two masks keep p_1 and filters out the superfluous zero p_2. *(c)* A resultant calculation of two polynomials, each implicitly defining the blue and green curves, yields a univariate polynomial with two real roots, r_1 and r_2; in particular, r_2 corresponds to a crossing point located outside the domain of interest. *(d)* Similar to the left example, mask computations effectively filter out the superfluous root r_2.

3.4.4. Application to the quadrature algorithm

Having established these tools to compute multivariate resultants and pseudo-discriminants, we are now in a position to describe their use in the quadrature algorithm.

First, we consider the points $\mathcal{R}$ in Fig. 1, i.e., the base points above which the zero level sets of two polynomials ϕ_i and ϕ_j intersect. To capture the location of these points, we can make use of the resultant of ϕ_i and ϕ_j in the height direction e_k. In particular, given a base point $x \in (0,1)^{d-1}$, let $\phi_{i,x}$ denote the univariate polynomial corresponding to the vertical line restriction of ϕ_i at x, i.e., so that $\phi_{i,x}(y) = \phi_i(x + ye_k)$ for all y, and similarly for ϕ_j. Then, $\big[R(\phi_i, \phi_j; k)\big](x) = 0$ if and only if $\phi_{i,x}$ and $\phi_{j,x}$ share a common root (real or complex). Consequently, the set of points $\mathcal{R}$ corresponding to the interfacial intersection of ϕ_i and ϕ_j is a subset of the zero isosurface of the polynomial $R(\phi_i, \phi_j; k)$.

A similar observation holds for the base points above which interfacial surfaces branch, i.e., where the multi-valued height function has a double root (points $\mathcal{P}$ in Fig. 1). We can capture the location of these points by making use of the pseudo-discriminant of ϕ_i. In particular, $\big[\Delta(\phi_i; k)\big](x) = 0$ if and only if $\phi_{i,x}$ and its derivative share a common root (real or complex). As such, the set of $\mathcal{P}$ points corresponding to ϕ_i is a subset of the zero isosurface of the polynomial $\Delta(\phi_i; k)$.

Note that, in both cases, the sets $\mathcal{R}$ and $\mathcal{P}$ may be proper subsets of the zero level sets of the respective resultant and pseudo-discriminant calculations. The reason for this is that resultant methods detect any shared roots, whether real or complex, including shared real roots that are outside the hyperrectangle of main interest to the quadrature algorithm. Some examples are shown in Fig. 2. In Fig. 2(a), the (pseudo-)discriminant calculation for a rotated ellipse yields a polynomial which has two real roots in the interval $\{0 \leq x \leq 1\}$, but only one of these roots corresponds to a branching point of the ellipse inside the unit square, whereas the other root is "superfluous" in the sense it detects a branching point outside the square of interest. In Fig. 2(c), the resultant calculation of two intersecting curves correctly identifies the base point of one junction, but also produces a superfluous base point for a junction that occurs outside the square domain.

These superfluous zeros produced by resultant and pseudo-discriminant calculations do not affect the "correctness" of the quadrature algorithms, but they can cause unnecessary subdivision in subsequent one-dimensional quadrature calculations. To help mitigate these effects, and in many cases eliminate them altogether, we make use of the masking operations described in §3.3. Specifically, the routine `intersectionMask` is used to produce a mask whose subcells are marked "0" if it can be provably determined that the two input polynomials do not share any common roots. These masks are then associated with corresponding resultant or pseudo-discriminant polynomials and essentially state that any roots of these polynomials in a subcell marked "0" can be ignored. For the examples of Fig. 2, these masks are illustrated in Fig. 2(b,d), wherein the non-shaded subcells correspond to the disabled parts of the mask; a root of the 1D resultant or pseudo-discriminant in one of the non-shaded subcells is then ignored by the quadrature algorithm. These masking operations are also effective in 3D. Furthermore, they are often effective in proving the non-existence of any $\mathcal{R}$ or $\mathcal{P}$ points, in which case the associated resultant or pseudo-discriminant calculations can be avoided altogether.

In general, the above methods for computing resultants and pseudo-discriminants are sufficient for the vast majority of applications we have in mind for the quadrature algorithms developed in this work. In exceptional cases, or deliberately contrived situations, additional tools may be needed to fully realise these methods. For example, if the user-provided multivariate polynomial is the square of another polynomial, it may need to be made square-free by using GCD factoring methods.[6] It is possible to incorporate additional steps into Algorithm 1 to automatically handle these edge cases, however we have omitted these details here for the benefit of clarity.

---

[6] Note, however, implicitly defined geometry given by the square of a polynomial does not lead to a well-posed problem for quadrature purposes: perturbations of the squared polynomial can result in the entire removal of the interface.

3.5. Choosing a height direction

At each stage of the dimension reduction process, a height direction/elimination axis e_k must be chosen. So far, we have seen how this choice affects the geometry of the base integral's domain via characterisation of the points $\mathcal{P}$, $\mathcal{Q}$, and $\mathcal{R}$ illustrated in Fig. 1, whose calculation, in turn, can depend on resultant and pseudo-discriminant calculations, as discussed in §3.4. The choice of height direction also impacts the accuracy of the subsequent quadrature schemes. Some of these effects include:

(i) Resultant and pseudo-discriminant computations can produce high degree polynomials in the description of the base integral's domain, an effect which can potentially compound further down the dimension reduction process for 3D (and higher dimension) problems. As such, it can be beneficial to limit the application of resultant and/or pseudo-discriminant polynomials.

(ii) The polynomials representing upper- and lower-face restrictions, i.e., those which characterise the points $\mathcal{Q}$, do not increase the degree of the input polynomial. As such, their creation and manipulation have a relatively minor impact on the overall complexity. However, if some faces of the hyperrectangle of $(0,1)^d$ have fewer or no interfacial intersections compared to other faces, it is preferential to use the corresponding height direction, as this means the geometry characterising the base integral's domain could be simpler, or even trivial.

(iii) As will be seen in the results, the accuracy of the overall quadrature scheme benefits from an effective application of Gauss-Legendre quadrature; in turn, these schemes work best if they are applied to interfacial geometry characterised by smooth, well-behaved height functions. In particular, a height direction in which there are no vertical tangents in the interface, i.e., $\mathcal{P}$ is empty, is preferred.

These objectives are not always mutually compatible. For example, a direction which avoids vertical tangents may necessitate non-trivial upper or lower face restrictions. However, a clear trend found in this work is that the best quadrature results—in terms of both accuracy and efficiency of construction—are obtained if e_k is chosen to be in the direction "most" orthogonal to the interfacial geometry. This metric prioritises the objectives (i) and (iii) above, in that it avoids vertical tangents and pseudo-discriminant calculations and greatly increases the odds of attaining high accuracy through effective application of Gauss-Legendre quadrature.

This heuristic is implemented follows. Suppose we are given a list of polynomials, $\{\phi_i\}$. First, we determine (cheaply, using masking operations) whether there are any directions k such that the interface of every given polynomial is provably the graph of a (potentially multi-valued) height function devoid of vertical tangents/branches.[7] Let $\mathcal{K}$ denote the set of all such directions k; if $\mathcal{K}$ is nonempty, then only those directions shall be considered in subsequent steps; otherwise, we set $\mathcal{K} = \{1, \ldots, d\}$ and consider all directions.

The height direction in Algorithm 1 on line 4 is then defined by choosing $k \in \mathcal{K}$ with the best "score", as follows (if $\mathcal{K}$ is a singleton, this step is trivial). For each i, let $m_i \in \mathbb{N}_{\mathcal{M}}^d \to \{0,1\}$ be the mask indicating which subcells of ϕ_i are provably nonzero (as computed by, e.g., nonzeroMask, see §3.3). Then the score in direction k is evaluated by

$$s_k = \sum_i \sum_{\substack{\ell \in \mathbb{N}_{\mathcal{M}}^d \\ m_i(\ell) \neq 0}} \frac{|\partial_k \phi_i(x_\ell)|}{\|\nabla \phi_i(x_\ell)\|_1}$$

where x_ℓ is the midpoint of subcell ℓ. In essence, this metric samples the gradient of ϕ_i at the midpoint of every mask subcell which (likely) contains the interface, weighted so that polynomials ϕ_i having greater interfacial measure have a greater influence in the overall score. In this fashion, the direction $k \in \mathcal{K}$ having largest score s_k heuristically determines which coordinate direction is most orthogonal to the combined interface(s) of every ϕ_i. Importantly, these mask operations and score evaluations are cheap and have negligible cost in the overall quadrature algorithm.

3.6. One-dimensional quadrature

One of the last remaining design aspects of the quadrature algorithm concerns the one-dimensional vertical line integrals, i.e., the choice of quadrature schemes in the implementation of Algorithm 2. Recall these line integrals correspond to evaluating, for a given base point $x \in (0,1)^{d-1}$, the base integral's integrand $\mathcal{F}(x)$, where $\mathcal{F}(x) = \oint_{(0,1)\backslash\Gamma(x)} f(x + y e_k)\, dy$, with f being the inner integral's integrand, while $\Gamma(x) = \bigcup_{i=1}^n \{y : 0 \leq y \leq 1 \text{ and } \phi_i(x + y e_k) = 0\}$ specifies the set of interface points above x.

To approximate $\mathcal{F}(x)$ for a given x, we first use root finding algorithms to compute the set of interfacial points $\Gamma(x)$. (Fast and backward stable methods for doing so are discussed in the supplementary material.) These roots are then used to partition $[0,1]$ into a set of one or more subintervals, after which a one-dimensional quadrature scheme is applied to each

---

[7] To do so, we can apply the intersectionMask routine to the polynomials ϕ_i and $\partial_k \phi_i$, and test whether the resulting mask is empty, i.e., identically 0-valued.

subinterval. With the goal of obtaining a simple "black box" quadrature algorithm, we here make two highly-simplifying design choices:

(i) Every subinterval shall apply the same kind of quadrature scheme, with the same number of quadrature points q per subinterval; and

(ii) The choice of quadrature scheme, whether it be Gauss-Legendre or tanh-sinh or an alternative, shall be independent of the base point x; in other words, the choice is made on a per-dimension basis.

Clearly, this approach is not at all adaptive; e.g., choosing q fixed can lead to an unnecessary number of quadrature points for small subintervals. The approach is also somewhat blind to the interfacial geometry at hand: different parts of Γ may exhibit different kinds of geometry and thus benefit from different kinds of quadrature that could vary throughout the domain of the base integral; as such, the universal scheme required by (ii) above may not always yield the smallest errors. Our experiments show, however, that the above simplifications nevertheless result in highly accurate quadrature schemes which exhibit exponential convergence in q. Additional remarks about these choices, including the possibility of adaptive methods, are given in the concluding remarks.

In our experiments, we have mainly explored the use of Gauss-Legendre and tanh-sinh quadrature schemes. The application of Gauss-Legendre methods (see, e.g., [25]) should be familiar to most readers and is not discussed further here; the use of tanh-sinh, however, is worthy of additional motivation. This scheme, also known as the double-exponential method, was originally developed by Takahasi and Mori [61] and makes use of the transformation $x = \tanh(\frac{1}{2}\pi \sinh(\xi))$ to map the interval $x \in (-1, 1)$ to the whole real line $\xi \in (-\infty, \infty)$ where then a trapezoidal rule is applied. The method can be highly effective at handling endpoint singularities, where in many cases it can achieve (near) exponential convergence. In the present work, our motivation for its application is to treat the cases of vertical tangents in the multi-valued height function; these vertical tangents give rise to an endpoint singularity in the base integral's integrand $\mathcal{F}$. As an example, consider the area integral of f on the upper half circle with height function $h(x) = \sqrt{1 - x^2}$; the task of the base integral is to then compute $\int_{-1}^{1} \left(\int_0^{h(x)} f(x, y) \, dy \right) dx$; for $f \equiv 1$, in particular, this reduces to $\int_{-1}^{1} \sqrt{1 - x^2} \, dx$. Applying Gauss-Legendre to this integral yields a poor convergence rate as q increases; on the other hand, tanh-sinh yields rapid, near exponential convergence [65].

Relative to its standard reference domain, $(-1, 1)$, the tanh-sinh quadrature method yields a quadrature scheme of the form $\int_{-1}^{1} f(x) \, dx \approx \sum_\ell w_\ell f(x_\ell)$; the standard tanh-sinh schemes usually have an odd number of quadrature points and do not exactly integrate constant functions, i.e., $\sum_\ell w_\ell \neq 2$. In the present work, we have implemented small modifications to allow for an even number of quadrature points and normalise the weights so that they exactly integrate constant functions[8]; importantly, these modifications do not alter the exponential convergence of these methods. Details are given in the supplementary material.

To summarise, Algorithm 2 approximates the vertical line integrals $\mathcal{F}(x)$ in the dimension reduction quadrature algorithm. Each subinterval uses, for simplicity, the same number of quadrature points q; each dimension shall apply, for simplicity, a fixed choice of Gauss-Legendre or tanh-sinh quadrature schemes, independent of the argument x. As such, the user may choose q and may choose Gauss-Legendre or tanh-sinh for each dimension of the reduction process. These choices, particularly of whether to apply Gauss-Legendre or tanh-sinh as a function of the detected interfacial geometry, is one of the main topics of discussion for the experimental results presented in §4.

3.7. Surface integration

In the formulation so far, we have only considered the case of volumetric integrals over implicitly defined domains. In essence, the same dimension reduction algorithm can be used in the case of surface integrals—the main modification needed is to replace the inner-most vertical line integral (corresponding to the very first elimination axis) with a new functional which evaluates the user-provided integrand f at the surface, appropriately weighted by Jacobian factors to account for its curvature. However, in some cases of interfacial geometry, particularly degenerate cases, it is important to appropriately handle singular Jacobian factors; here, we discuss an approach which robustly handles these situations.

The problem set up is as follows. Suppose we are given one or more polynomial functions $\phi_i : (0, 1)^d \to \mathbb{R}$, $i = 1, \ldots, n$, which together implicitly define an interface $\Gamma := \bigcup_{i=1}^{n} \{\phi_i = 0\}$. Suppose also that we are given an integrand $f : \Gamma \to \mathbb{R}$, which need only be defined on the interface Γ. Our task is to build a high order quadrature scheme approximating $\int_\Gamma f$. Applying the same dimension reduction ideas as in §3.1, suppose we have chosen a height direction, e_k, say. Then, we may reinterpret Γ as the graph of a (potentially multi-valued) height function and compute the surface integral correspondingly. The curvature of the height function induces a surface Jacobian factor: in [51] it is shown (via the implicit function theorem) this can be computed via the gradient of the associated level set function. Using these ideas, we obtain $\int_\Gamma f = \int_\Omega \mathcal{F}(x) \, dx$ where Ω is the domain of the base integral, $x \in \Omega \subseteq (0, 1)^{d-1}$ is a base point, and

---

[8] This normalisation results in the guarantee that any quadrature scheme produced by the dimension reduction algorithm satisfies the condition $\int_{(0,1)^d \setminus \Gamma} 1 = 1$.

$$\mathcal{F}(x) = \sum_{i=1}^{n} \sum_{y \in \Gamma_i(x)} f(x + y e_k) \left. \frac{|\nabla \phi_i|}{|\partial_k \phi_i|} \right|_{x + y e_k} \tag{8}$$

where

$$\Gamma_i(x) := \{y : 0 \le y \le 1 \text{ and } \phi_i(x + y e_k) = 0\}.$$

In essence, the base integral's integrand $\mathcal{F}(x)$ evaluates the given integrand f at every interfacial point above x; at each such point, the curvature of the interface is accounted for by the Jacobian factor $|\nabla \phi|/|\partial_k \phi|$.

If there is a height direction e_k such that every such Jacobian factor is well-defined, we can then employ the existing dimension reduction algorithm—in doing so, a quadrature scheme for $\int_\Gamma f = \oint_\Omega \mathcal{F}(x)\,dx$ is constructed, with Ω defined in the same way as the volumetric case, i.e., as the set of points $(0,1)^{d-1} \setminus (\mathcal{Q} \cup \mathcal{R})$ in which (see Fig. 1) the upper- and lower-face interfacial crossings $\mathcal{Q}$ and internal junctions $\mathcal{R}$ have been removed (there are no points $\mathcal{P}$ by assumption). However, this approach cannot be applied if no such height direction exists.

Two examples help to illustrate the situation at hand, the second more severe than the first:

(i) Consider ϕ given by $\phi(x, y) = (x - \frac{1}{2})^2 + (y - \frac{1}{2})^2 - \frac{1}{16}$, whose interface is a circle of radius 0.25 centred at $(0.5, 0.5)$. Then, every elimination axis yields a height function with vertical tangents; e.g., for $e_k = \hat{y}$, the height function has a vertical tangent at $x \in \{0.25, 0.75\}$. At these locations, the surface Jacobian factor $|\nabla \phi|/|\partial_y \phi|$ is not well-defined. One potential workaround is to apply tanh-sinh quadrature in the base integral and potentially yield exponential convergence[9]; however, in more general cases it can be problematic to yield full machine precision due to the way tanh-sinh sends quadrature points exponentially close to the integral endpoints, combined with roundoff errors in computing polynomial roots.

(ii) Consider ϕ given by $\phi(x, y) = (x - \frac{1}{2})(y - \frac{1}{2})$, whose interface is a cross of the form ✚ centred at $(0.5, 0.5)$. The approach used by (8) is not all amenable to this example. For example, with elimination axis $\hat{y}$, the dimension reduction algorithm will correctly sense the location of the vertical arm of the interface, but the associated integrand $\mathcal{F}(x)$ given by (8) at $x = 0$ is not at all well-defined. There is no means for the base integral to suitably place quadrature points on the vertical arm of the interface and to assign well-posed quadrature weights. However, the base integral can correctly handle the horizontal arm—this gives a hint on how to handle the general case, which is to apply the dimension reduction algorithm in all dimensions e_k, $k = 1, \ldots, d$, and then aggregate the results.

Exploring this idea further, note that the surface Jacobian factor $|\nabla \phi|/|\partial_y \phi|$ in (8) is precisely $1/|n_k|$, where $\mathbf{n}$ is the unit-magnitude normal to the interface Γ. As such, performing a surface integral, not of f, but of an integrand of the form $f n_k$, will cancel the problematic term. For such an integrand, $\mathcal{F}$ now takes the form

$$\mathcal{F}(x) = \sum_{i=1}^{n} \sum_{y \in \Gamma_i(x)} f(x + y e_k) \, \text{sign}\big(\partial_k \phi_i\big)\big|_{x + y e_k}. \tag{9}$$

This integrand is piecewise smooth on the connected components of the base integral's domain $\Omega = (0,1)^{d-1} \setminus \{\mathcal{P} \cup \mathcal{Q} \cup \mathcal{R}\}$; in particular the sign term is constant on each connected component due to the removal of the set $\mathcal{P}$ (recall that $\mathcal{P}$ characterises the base points above which the zero isosurfaces of ϕ_i and $\partial_k \phi_i$ intersect). As such, we can use the existing dimension reduction algorithm to build a high order quadrature scheme for $\int_\Gamma f n_k = \oint_\Omega \mathcal{F}(x)\,dx$. Performing this process for every height direction $k = 1, \ldots, d$ then yields a robust and accurate means to compute a quadrature scheme of $\int_\Gamma f \mathbf{n}$, for all sufficiently smooth f.

In summary, the above considerations suggest the following two mechanisms:

- If it can be proven that the interface is the graph of a (multi-valued) height function in direction e_k, without vertical tangents, and such that the associated Jacobian factors $|\nabla \phi|/|\partial_k \phi|$ are sufficiently well-behaved, then we can apply Algorithm 1 where line 23 is replaced with a function evaluating (8).
- Otherwise, apply an "aggregated" method. For each $k = 1, \ldots, d$, execute Algorithm 1 with the following modifications: (i) overrule line 4 to use the height direction k; and (ii) replace line 23 with a suitable functional evaluating (9). For each k, the output of this process is a quadrature scheme approximating $\int_\Gamma f n_k$.

Clearly, the aggregated method is computationally more expensive than the first approach, having to iterate over each possible choice of elimination axis. However it is more well-posed and less sensitive to perturbations in the input geometry.

---

[9] One can show that, on the upper half of the circle with corresponding height function $h(x)$, the base integral's integrand takes the form $\mathcal{F}(x) = f(x, h(x))\sqrt{1 + (h'(x))^2}$, where the latter term has a singularity of the form $1/\sqrt{\tilde{x}}$ with $\tilde{x}$ being the distance to the circle endpoints. A square-root singularity like this can be easily handled by tanh-sinh quadrature, provided certain measures are taken to guard against floating point roundoff errors [2,67].

Note that the aggregated method essentially produces a surface quadrature scheme in *flux form*, i.e., given a sufficiently smooth function f, the algorithm produces a quadrature scheme $\int_\Gamma f\mathbf{n} \approx \sum_\ell \mathbf{w}_\ell f(x_\ell)$ where $\mathbf{w}_\ell$ are vector-valued quadrature weights. If needed, one could then build a surface quadrature scheme in *non-flux form* by baking in the normal vector to yield a scheme of the form $\int_\Gamma g \approx \sum_\ell (\mathbf{w}_\ell \cdot \mathbf{n}_\ell) g(x_\ell)$, where $\mathbf{n}_\ell$ is the normal vector of the interface associated with x_ℓ. Note that in many applications of surface quadrature, the flux form of the surface integral is the most relevant or natural. For example, in finite volume methods, discontinuous Galerkin methods, etc., the associated weak formulations require elemental boundary integrals which are typically in flux form, e.g., to compute $\int_\Gamma \hat{f}\mathbf{n}$ where $\hat{f}$ is a numerical flux. Our numerical experiments (see §4) show that the flux-form of surface integration generally yields much higher accuracy than the non-flux form, essentially because the normal component n_k exactly cancels the surface Jacobian factors $1/|n_k|$.[10]

3.8. Summary

In essence, the quadrature algorithm operates by recasting the implicitly defined interface as the graph of a multi-valued height function. Using the corresponding splitting of coordinates, the overall integral is reformulated as a base integral with an associated integrand $\mathcal{F}$. In the case of volumetric integrals, the role of $\mathcal{F}$ is to compute a vertical line integral in the height direction; in the case of surface integrals, the role of $\mathcal{F}$ is to evaluate the user-provided integrand on the interface Γ, appropriately modified by curved surface Jacobian factors. Continuing recursively, we are led to a conceptually simple dimension reduction algorithm—in order to achieve a high level of accuracy, it is important to appropriately identify and treat the piecewise smooth behaviour of $\mathcal{F}$. In turn, this corresponds to identifying the different characteristics of the height function, notably where Γ crosses the upper- and lower-faces, its branching points, and the intersection points. The latter two require some workhorse tools from algebraic geometry, i.e., resultants and (pseudo-)discriminants, and are perhaps the most complex components of the quadrature algorithm. In §3.4.4, we saw that these resultant methods can sometimes produce superfluous zeros, some of which correspond to branching/junction points outside the unit cube of interest. Many of these superfluous zeros can be removed by the masking methods. The masks also provide additional means for (cheaply) characterising the interface geometry, and we use them to help decide which coordinate axis to use as a height direction, as well as guide decisions on which kind of one-dimensional quadrature to use.

These ideas led to the main driver algorithms, Algorithm 1 and Algorithm 2. Starting with the top-level integral problem, a coordinate axis is chosen for the height direction, and then a set of polynomials is constructed to determine the implicitly defined domain of the base integral. This process continues recursively, eliminating one coordinate axis at a time, down to the one-dimensional base case. The base case then performs a one-dimensional quadrature and this process continues recursively back up the tree: each integrand evaluation adds one more dimension at a time, terminating with the evaluation of the user-supplied top-level integrand f. In particular:

- To compute a volumetric quadrature scheme corresponding to

$$\oint_{(0,1)^d \setminus \Gamma} f \approx \sum_i w_i f(x_i),$$

  where $\Gamma = \bigcup_{i=1}^n \{\phi_i = 0\}$ is implicitly defined by $n \geq 1$ user-defined polynomials ϕ_i, Algorithm 1 is executed with the integrand f and every mask initialised by $m_i \equiv 1$. These mask values can be viewed as a simple default, conveying that any part of $(0,1)^d$ may contain the interface. The output is a quadrature scheme whose weights are, by construction, strictly positive. In fact, the nodes fill the unit cube and by normalisation conditions, we have $\sum_i w_i = 1$. Importantly, the scheme is accurate on each connected component of $(0,1)^d \setminus \Gamma$. As such, the sign of ϕ_i can be used to cluster the quadrature nodes; for example, to compute

$$\int_{(0,1)^d \cap \bigcap_{i=1}^n \{\phi_i < 0\}} f,$$

  one can keep just the quadrature points satisfying $\phi(x_i) < 0$, leaving their weights unmodified. We note the algorithms can be easily adjusted if it was known ahead of time that only one side of Γ is needed.
- To compute a surface quadrature scheme, either for $\oint_{(0,1)^d \cap \Gamma} g$ or for $\oint_{(0,1)^d \cap \Gamma} g\mathbf{n}$, follow the advice of §3.7. The first is a surface integral in non-flux form, whereas the second is in flux-form and typically has higher accuracy. By construction, the resulting quadrature nodes are strictly on $(0,1)^d \cap \Gamma$. A similar operation as above could be used to cluster the nodes according to the different parts of a multi-polynomial interface.

Fig. 3 illustrates the output of the quadrature algorithm for various kinds of interfacial geometry in 2D and 3D, with a single input polynomial (multiple polynomials are demonstrated later in §4.7.2). In each case, the quadrature scheme

---

[10] For example, in two dimensions, the flux-form integral of $f \equiv 1$, over the one-dimensional curve Γ, is computed exactly, independent of q.
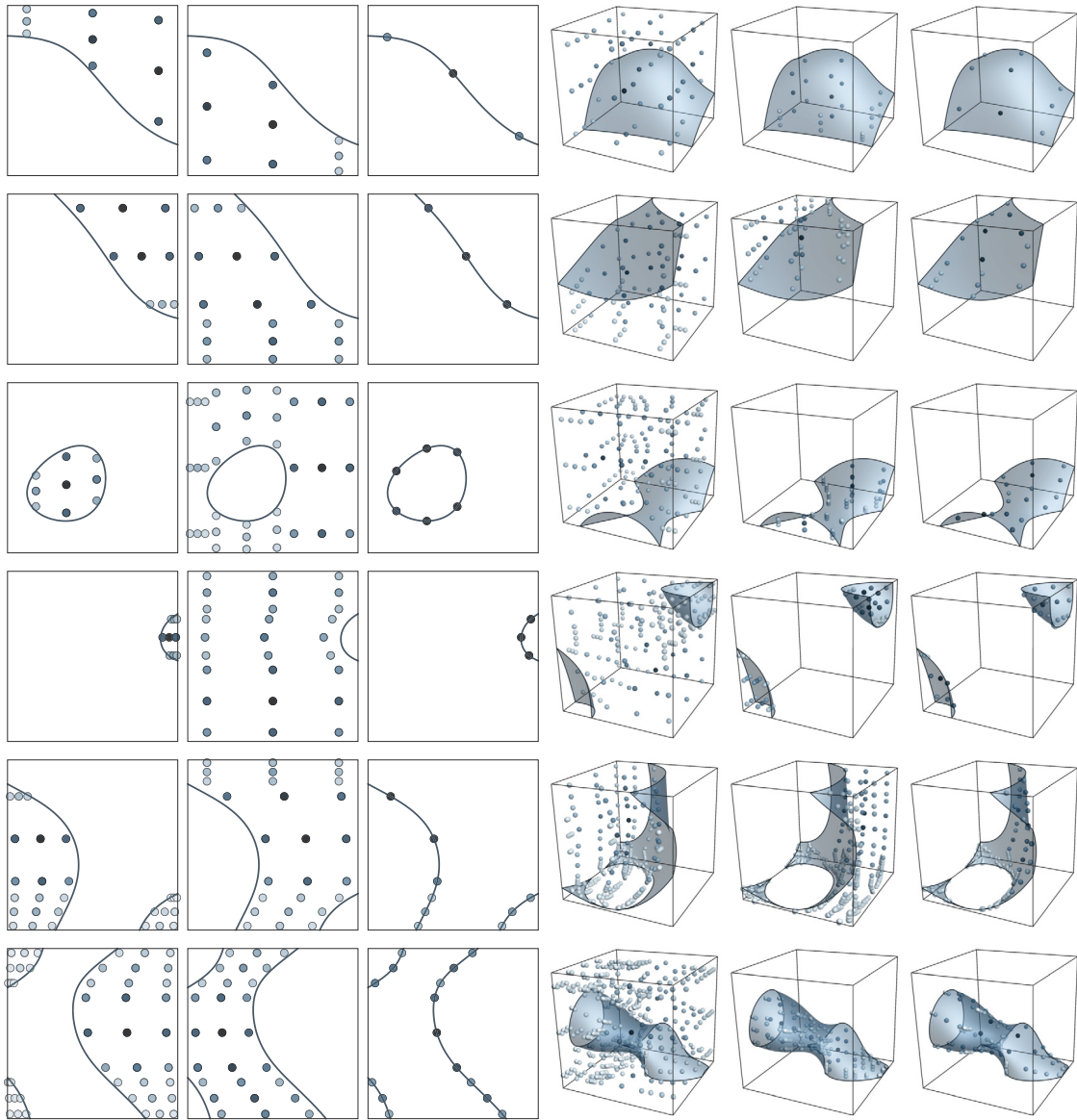
Fig. 3. Examples illustrating the output of the quadrature algorithm for different kinds of interfacial geometry, using $q = 3$ quadrature nodes per one-dimensional integral. In each group of three, the left and middle figures correspond to volumetric integrals over the implicitly defined regions $\{\phi < 0\}$ and $\{\phi > 0\}$, while the right figure corresponds to the surface integral over $\{\phi = 0\}$; in each instance, the quadrature points are shaded according to their weight, such that low-valued weights are pale and higher-valued weights are darker.

consists of one or more patches of points in tensor-product form, with each patch warping to the geometry of the interface; this is most easily observed in the 2D examples and some of the simpler 3D examples. Naturally, the more complex 3D examples require more patches of points.

4. Numerical experiments

In this section, we demonstrate the quadrature algorithm and investigate its accuracy through a variety of numerical experiments. The test problems range from interfacial geometry of mild to modest difficulty, yielding high-order convergence rates, through to more challenging scenarios involving cusps, pushing the limits of what can be achieved in finite precision arithmetic. In general, the quadrature schemes yield high degrees of accuracy, e.g., approximately exponential convergence in the number of quadrature points. In some cases, convergence to machine precision is quickly attained and this can prevent a clear observation of asymptotic convergence rates. To help with these effects, in some of our presented results we have made use of high-precision arithmetic via the QD library [3], which provides double-double and quadruple-double floating
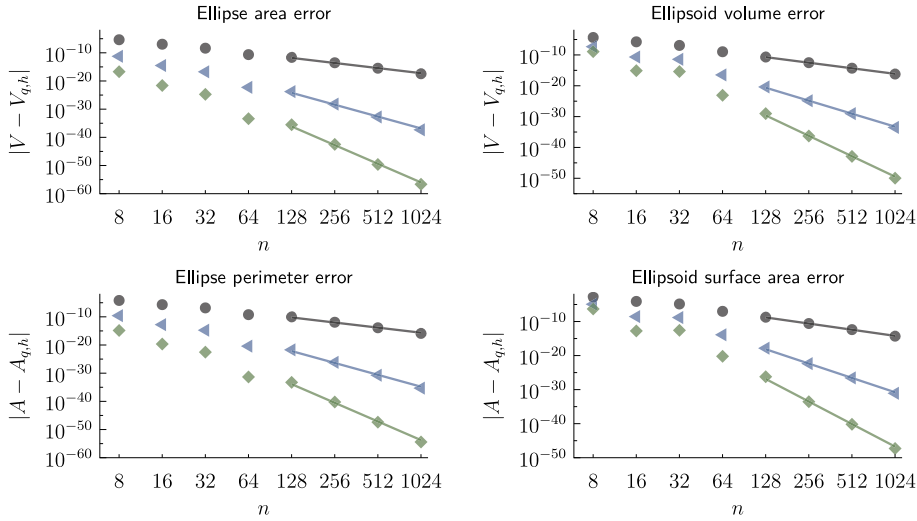
Fig. 4. Results of the ellipsoidal test problems considered in §4.1, studying quadrature accuracy under the action of h-refinement, for three different choices of quadrature order q: $\bullet$ (black) denotes $q = 3$ $\blacktriangleleft$ (blue) denotes $q = 7$, and $\blacklozenge$ (green) denotes $q = 11$; the corresponding lines have slope $2q$, indicating the $\mathcal{O}(h^{2q})$ asymptotic convergence rate as $h = 2.2/n \to 0$.

point routines. We emphasize that high-precision arithmetic is not a requirement of the quadrature algorithms—these tools are only being used to help reveal the asymptotic convergence behaviour and quantify errors in relation to machine epsilon.

4.1. Convergence under h-refinement

The majority of the presented results consider a fixed constraint cell U and examine convergence under the action of q-refinement, i.e., as more and more quadrature points are used. In this study, however, we consider convergence under h-refinement, wherein both q and the interfacial geometry is globally fixed and the cell U is made smaller and smaller. Under this kind of refinement, for a globally smooth interface, the local interfacial geometry, $U \cap \Gamma$, becomes asymptotically flat; as such, we expect high rates of convergence, matching those of a Gaussian quadrature scheme. To confirm this, we consider a test problem of computing the surface area and volume of a d-dimensional ellipsoid:

- In 2D, the ellipse is given by the zero level set of $\phi(x, y) = x^2 + 4y^2 - 1$; the ellipse has semimajor axis 1, semiminor axis $\frac{1}{2}$, an area of $V = \frac{\pi}{2}$, and perimeter $A = 4.84422411\ldots$.
- In 3D, the ellipsoid is given by the zero level set of $\phi(x, y, z) = x^2 + 4y^2 + 9z^2 - 1$; the ellipsoid has semiprincipal axes 1, $\frac{1}{2}$, and $\frac{1}{3}$, a volume of $V = \frac{2\pi}{9}$, and surface area $A = 4.40080956\ldots$.

We consider a reference domain $(-1.1, 1.1)^d$ enclosing the geometry, discretised by a uniform Cartesian grid consisting of n cells in each direction, such that $h = 2.2/n$. For a fixed q, the quadrature algorithm is then applied to each cell, and the results summed over the entire grid so as to compute the total surface area and volume, denoted $A_{q,h}$ and $V_{q,h}$, respectively. In particular, and for simplicity of discussion, in these results we use Gauss-Legendre quadrature for all of the one-dimensional integrals executed by Algorithm 2.

In the presented results, we have made use of quadruple-double precision so as to more clearly reveal the asymptotic high-order convergence rates. Fig. 4 plots the errors in the computed surface area and volume of the ellipse and ellipsoid, for three different values of $q \in \{3, 7, 11\}$. In each case, the fitted line of slope $2q$ confirms that the quadrature schemes inherit the high-order convergence rate of Gaussian quadrature. For example, 22nd-order convergence rates are obtained with $q = 11$, on all four test problems. Similar results are obtained for other values of q: we have tested $q = 1, 2, \ldots, 11$ and observed $\mathcal{O}(h^{2q})$ errors in every case. Besides the ellipsoidal geometry of this example, the same conclusions hold as well for other kinds of sufficiently smooth geometry and integrand functionals.

4.2. Ellipsoid under q-refinement

Here we consider the same ellipsoidal geometry as in the previous problem, and examine quadrature accuracy under the action of q-refinement, i.e., keeping the geometry $U \cap \Gamma$ fixed while the quadrature order q varies. In particular, we examine the error in computing the following three integral quantities,

$$I_\Omega := \int_{U \cap \{\phi < 0\}} f, \quad I_\Gamma := \int_{U \cap \{\phi = 0\}} f, \quad I_{\Gamma_{\mathbf{n}}} := \int_{U \cap \{\phi = 0\}} f\mathbf{n},$$
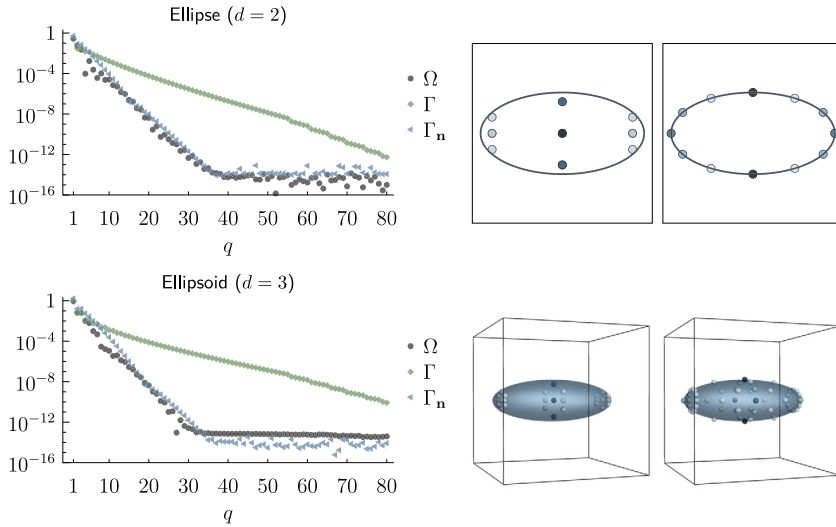
Fig. 5. *(left)* Quadrature accuracy as a function of q for the ellipsoidal test problems considered in §4.2. Here, • (black) denotes the relative error in the computed volumetric integral, I_Ω; ◆ (green) denotes the relative error in the computed surface integral in non-flux form, I_Γ; and ◀ (blue) denotes the relative error in the computed flux-form surface integral, $I_{\Gamma_\mathbf{n}}$. *(right pair)* Illustrations of the $q = 3$ volumetric and (aggregated) surface quadrature schemes; in each instance, the quadrature points are shaded according to their weight, such that low-valued weights are pale and higher-valued weights are darker.

where $U := (-1.1, 1.1)^d$ is fixed, and the integrand $f : \mathbb{R}^d \to \mathbb{R}$ is given by[11] $f(x) = \cos(\frac{1}{4}\|x - \frac{1}{4}\|_2^2)$. Here, ϕ denotes the level set function defining the ellipse/ellipsoid (identical that defined in §4.1), I_Ω denotes the volumetric integral of f over the inside of the ellipse/ellipsoid, while I_Γ and $I_{\Gamma_\mathbf{n}}$ denote the surface integral of f in non-flux and flux-form, respectively. Using the strategy outlined in §3.5 for choosing the height direction, in computing the volumetric integral I_Ω the quadrature algorithm operates as follows:

- In 2D, the first elimination axis is $\hat{y}$, and so the outer integral is over the x-axis and the inner vertical line integrals occur in the $\hat{y}$ direction. The outer integral sees a height function with branching points (similar to Fig. 1, left); consequently, we apply tanh-sinh quadrature on the outer integral, and Gauss-Legendre quadrature on the inner integral.
- In 3D, the first elimination axis is $\hat{z}$, followed by elimination of the $\hat{y}$ axis, and so the outer-most integral is over x and the inner-most integral over z. Similar to Fig. 1, left, the outer integrals involve height functions with vertical tangents, and so we apply tanh-sinh quadrature in Algorithm 2 for integrals over the x and y direction, and apply Gauss-Legendre quadrature on the inner-most integral over z.

In both the 2D and 3D cases, the surface integrals are computed using the aggregated method discussed in §3.7. Illustrations of these schemes for $q = 3$ are shown in Fig. 5.

For each integral, I_Ω, I_Γ, and $I_{\Gamma_\mathbf{n}}$, a reference solution (accurate to full machine precision) is computed using a high degree quadrature order, specifically $q = 100$. We then examine the error in these integrals, compared to the reference solution, using quadrature orders ranging from $q = 1$ up to $q = 80$; Fig. 5 contains the results,[12] computed using native double precision. In the case of the volumetric integral and the flux-form surface integral, we observe rapid, approximately exponential convergence in q, with both integrals attaining maximum possible precision at around $q \approx 34 \pm 2$. This number is typical for tanh-sinh quadrature applied to integrals of the form $\int_{-1}^{1} \sqrt{1 - x^2}$. However, for the surface integral in non-flux form, I_Γ, we observe a slower convergence rate, albeit also exponential. This slower convergence in computing I_Γ, versus $I_{\Gamma_\mathbf{n}}$, confirms the discussion of §3.7; in particular, the method to compute I_Γ is essentially computing the surface integral of $f |n_k|^2$, with the result summed over all k, where $\mathbf{n}$ is the normal to the interface. The corresponding integrand is bounded and smooth, but its analytic extension to the complex plane has a pole close enough to the origin that it impacts the convergence rate of tanh-sinh quadrature. For example, in the 2D case, one can show that, for elimination axis $\hat{x}$, the y integral over $-\frac{1}{2} < y < \frac{1}{2}$ has an integrand of the form $f(h(y), y)\sqrt{1 - 4y^2}/\sqrt{1 + 12y^2}$; this latter integrand has a pole inside the unit disc of the complex plane, and it is well-known that convergence of tanh-sinh slows in this case [65]. In essence, the (aggregated) surface integral in non-flux form computes integrals with an extra factor of $|n_k|$ in the integrand (versus the flux-form), thereby changing its regularity, which in turn impacts the convergence rate. This phenomenon is observed in many of our results, and further comments are given in the concluding remarks.

---

[11] There is nothing particularly special about the chosen integrand f, other than being smooth, non-trivial, positive on U, and such that $I_{\Gamma_\mathbf{n}}$ is nonzero.

[12] This figure, and all others like it in this paper, plot the relative error compared to the reference/exact solution, e.g., $|I_{\Omega,q} - I_{\Omega,\text{exact}}|/|I_{\Omega,\text{exact}}|$. In the case of the flux-form surface integral, the vector maximum norm is used, i.e., $\|I_{\Gamma_\mathbf{n},q} - I_{\Gamma_\mathbf{n},\text{exact}}\|_\infty / \|I_{\Gamma_\mathbf{n},\text{exact}}\|_\infty$.

4.3. Randomly generated geometry

We next consider quadrature performance on a class of randomly generated polynomials whose corresponding implicitly defined geometry exhibits a wide variety of characteristics, such as multi-component domains, multi-valued height functions, high degrees of curvature, and tunnels. Through the computation of aggregated statistics, our goal is to: (i) help determine an automated strategy for choosing between Gauss-Legendre and tanh-sinh quadrature in the overall quadrature algorithm; and (ii) assess quadrature convergence on the kinds of interfacial geometry which may arise in a complex computational fluid dynamics problem, e.g., modelling atomizing liquid.

The class of randomly generated polynomials is defined through the orthonormal Legendre polynomials, as follows. Let p_i, $i = 0, 1, 2$, denote the first three normalised univariate Legendre polynomials relative to the interval $(-1, 1)$, i.e.,

$$p_0(x) = \sqrt{\tfrac{1}{2}}, \quad p_1(x) = \sqrt{\tfrac{3}{2}}\, x, \quad p_2(x) = \sqrt{\tfrac{5}{8}}(3x^2 - 1).$$

In d dimensions, define a tensor-product degree $(2, \ldots, 2)$ polynomial $\phi : \mathbb{R}^d \to \mathbb{R}$ as follows:

$$\phi(x) = \sum_{i \in \{0,1,2\}^d} \lambda(i)\, c_i \prod_{\ell=1}^{d} p_{i_\ell}(x_\ell), \tag{10}$$

where $c_i \in \mathbb{R}$ is a given set of 3^d coefficients, one for each multi-index $i \in \{0, 1, 2\}^d$, and $\lambda(i)$ is a scaling factor such that

$$\lambda(i) = \begin{cases} 1 & \text{if } i = (0, \ldots, 0), \\ \left(\sum_{\ell=1}^{d} i_\ell\right)^{-\alpha} & \text{otherwise.} \end{cases}$$

Here, $\alpha > 0$ is a decay factor controlling the smoothness of the polynomial ϕ via the damping of higher-order modes. With this set up, we define a class of randomly generated polynomials by constructing polynomials of the form (10), with coefficients c_i randomly and independently drawn from the uniform distribution on $[-1, 1]$. In particular, we choose $\alpha = 2$, empirically chosen to represent a reasonable spread between simple and smooth interfacial geometry, to complex geometry involving high degrees of curvature.[13] We further characterise the class as follows:

- Let $\mathcal{A}$ denote the class of polynomials defined by (10), with $\alpha = 2$ and c_i uniformly randomly chosen from the interval $[-1, 1]$, such that the following two conditions hold: (i) the volume fraction of $\{\phi < 0\}$ in $U = (-1, 1)^d$ is between 10% and 90%; and (ii) using masking operations, it can be determined there exists a direction e_k such that there are no branching points in the interface, i.e., ϕ and $\partial_k \phi$ do not share a real root in U.
- Let $\mathcal{B}$ be defined in the same way as $\mathcal{A}$, except that condition (ii) cannot be verified, i.e., masking operations determine that each direction e_k (likely) has a branching point in its corresponding implicitly defined height function, for every $1 \le k \le d$.

Thus, $\mathcal{A}$ and $\mathcal{B}$ represent two classes of randomly generated polynomials, differing according to the automated strategy of choosing a height direction discussed in §3.5—for polynomials in class $\mathcal{A}$, there is an elimination axis for which it can be provably determined that the associated implicitly defined (multi-valued) height function is devoid of branching points, whereas polynomials in class $\mathcal{B}$ are likely to have vertical tangents for every possible height direction. As such, the two classes roughly separate the "severity" of the implicitly defined geometry in its representation as an implicitly defined height function. Meanwhile, the volume fraction condition is simply used to eliminate cases in which $U \cap \Gamma$ is empty or nearly empty.

Examples of the randomly generated polynomials are illustrated in Fig. 6. One may observe that for each instance of class $\mathcal{A}$, there is a direction in which the corresponding height function has no vertical tangents, even if the height function is multi-valued. The examples of class $\mathcal{B}$ are generally more complex and contain vertical tangents for every possible elimination axis.

Next we define an error metric by which to assess the accuracy of the computed quadrature schemes on the above randomly generated geometries. Here we define a metric having close connection to cut-cell/embedded boundary finite element methods. Specifically, the quadrature algorithm is used to compute the mass matrix of a rectangular/prismatic element which has been cut by the implicitly defined geometry. Define the matrix-valued quantity $I^\pm$, whose (i, j)th entry is given by

$$I^\pm_{ij} = \int\limits_{U \cap \{\pm\phi < 0\}} f_i f_j, \tag{11}$$

---

[13] In particular, $\alpha \ge 3$ tends to bias towards interfacial geometry with mild curvature, $\alpha \le 1$ tends to solely create complex interfacial geometry, while $\alpha = 2$ "evenly" spreads between these characteristics.
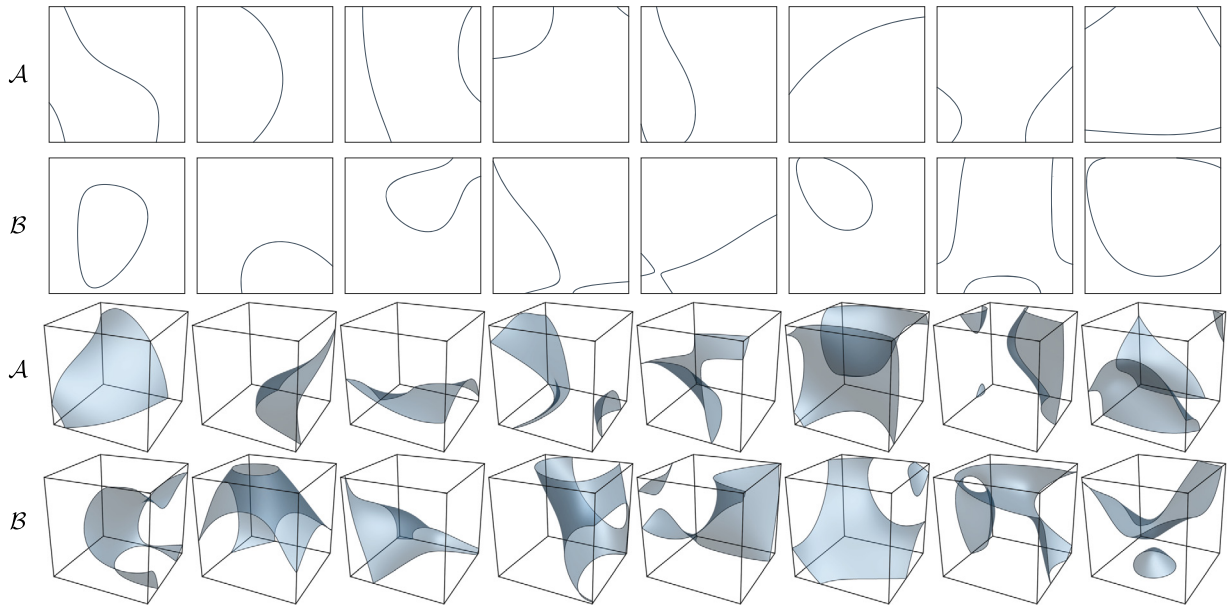
Fig. 6. Examples of randomly generated interfacial geometry, corresponding to "typical" instances of the randomly generated polynomials of classes $\mathcal{A}$ and $\mathcal{B}$ defined in §4.3. The top and bottom two rows correspond to $d = 2$ and $d = 3$, respectively.

where the functions f_i enumerate the polynomials of the Gauss-Lobatto nodal basis of biquadratic ($d = 2$) or triquadratic ($d = 3$) polynomials on $U = (-1, 1)^d$; for details, see, e.g., [48]. Therefore, I^+ computes the mass matrix of the cut element on the positive side of the interface $\Gamma = \{\phi = 0\}$, and I^- computes the mass matrix on the negative side. Let $I_q^\pm$ denote the mass matrix computed by the quadrature scheme using a quadrature order q, and let $I_{\text{ref}}^\pm$ denote a reference solution.[14] A relative error E_q is defined such that

$$E_q = \frac{\|I_q^+ - I_{\text{ref}}^+\|_F + \|I_q^- - I_{\text{ref}}^-\|_F}{\|I_{\text{ref}}^+\|_F + \|I_{\text{ref}}^-\|_F}, \tag{12}$$

where $\|\cdot\|_F$ is the Frobenius norm. As such, E_q measures the relative error in the computed mass matrices of both phases, $U \cap \{\phi < 0\}$ and $U \cap \{\phi > 0\}$.

The mass matrix error metric is used to examine the effectiveness of the quadrature algorithm on randomly generated geometry, as follows:

- One thousand instances[15] of the polynomials in classes $\mathcal{A}$ and $\mathcal{B}$ are randomly generated (2000 total in 2D and 2000 in 3D). In every case, the automated strategy discussed in §3.5 is used to determine the height direction, at every level of the dimension reduction algorithm.
- The kind of quadrature scheme to use in Algorithm 2, whether it is Gauss-Legendre or tanh-sinh, is defined *a priori*, and shall depend only on the level of the dimension reduction process. Specifically, in the 2D case, the notation (b, a) means that quadrature of type b is used on the outer integral, and a on the inner integral; in the 3D case, the notation (c, b, a) means quadrature of type c is used on the outer-most integral, a on the inner-most integral of the first eliminated axis, and b on the second eliminated axis. In particular, the objective here is to compute the entries of a mass matrix for which the integrand of the inner-most integral is the product of two polynomials, see (11); consequently, a natural choice is to apply Gauss-Legendre on the inner-most integral,[16] and so a = GL in the following tests.
- For each polynomial and each combination of quadrature scheme type, the relative error E_q is computed using quadrature orders ranging from $q = 1$ up to $q = 80$. In general, these quantities will decrease exponentially, up to the limits of machine precision, with a rate depending on the specific polynomial. To examine the corresponding distribution of

---

[14] In the following results, the reference solution is computed using a large number of quadrature points, specifically $q = 100$ per sub-interval in Algorithm 2.

[15] Experiments indicated that only a few hundred instances are needed to stabilise the computed statistics.

[16] When $q \geq 3$, Gauss-Legendre quadrature will exactly integrate the inner-most vertical line integrals, the latter involving the product of two quadratic polynomials. Tanh-sinh, although rapidly convergent, will not be exact for such integrands.
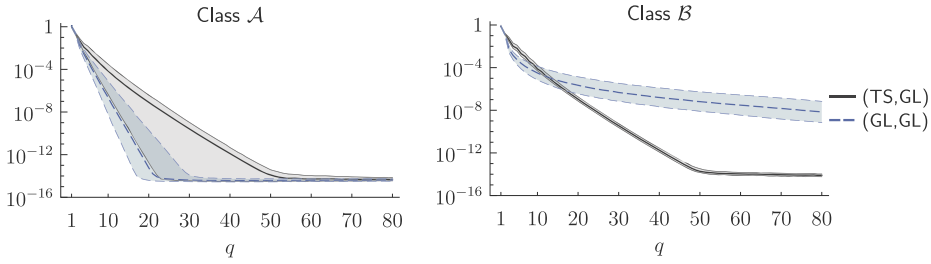
Fig. 7. Quadrature accuracy as a function of q for the two-dimensional randomly generated interfaces considered in §4.3; some examples are shown in Fig. 6. The quadrature combination (GL, GL) is indicated in blue with dashed lines, whereas the combination (TS, GL) is indicated in grey with solid lines. In each case, the shaded region demarcates the boundaries of the first and third quartiles of E_q, defined by (12), aggregated over all 1000 polynomial instances, while the interior line indicates the median.
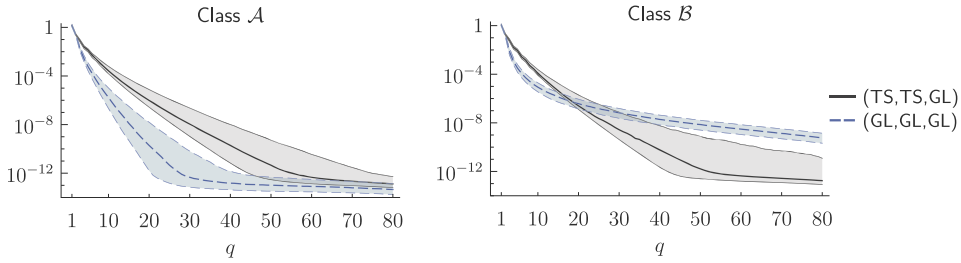
Fig. 8. Quadrature accuracy as a function of q for the three-dimensional randomly generated interfaces considered in §4.3; some examples are shown in Fig. 6. The quadrature combination (GL, GL, GL) is indicated in blue with dashed lines, whereas the combination (TS, TS, GL) is indicated in grey with solid lines. In each case, the shaded region demarcates the boundaries of the first and third quartiles of E_q, defined by (12), aggregated over all 1000 polynomial instances, while the interior line indicates the median.

convergence rates, we examine, as a function of q, the first, second, and third quartiles[17] of E_q, aggregated over all randomly generated polynomials in each particular class.

4.3.1. Two-dimensional results

In 2D, we consider two combinations of quadrature schemes, (GL, GL) and (TS, GL). Fig. 7 illustrates the results of the study, divided according to the classes $\mathcal{A}$ and $\mathcal{B}$. Several observations can be made:

- For the randomly generated geometry of class $\mathcal{A}$, the quadrature accuracy using (GL, GL) is generally far superior to (TS, GL). Indeed, the corresponding interfacial geometry is the graph of a smooth height function devoid of branches, and as such we intuitively expect to benefit from the rapid convergence of high-order Gaussian quadrature. Moreover, when Gaussian quadrature functions effectively, it can be expected to yield more rapid convergence than tanh-sinh quadrature.
- For polynomials in class $\mathcal{B}$, the quadrature choice (GL, GL) performs very poorly, and does not reach machine precision. This should also be intuitively expected, as Gaussian quadrature performs poorly on integrands with endpoint singularities; in this case, the endpoint singularities occur in the derivative of the integrands, corresponding to vertical tangents of the height function. On the other hand, (TS, GL) is effective at treating these kinds of singularities. However, note that for small q, around ten or less, (GL, GL) can be substantially more accurate than (TS, GL); in a sense, tanh-sinh quadrature can require a moderate number of quadrature points before its convergence accuracy is ultimately superior to Gaussian quadrature.
- One can also observe that the spread in errors for (TS, GL) on class $\mathcal{B}$ is quite small; evidently, this combination of quadrature scheme behaves consistently across the wide range of possible interfacial geometry in class $\mathcal{B}$.

4.3.2. Three-dimensional results

In 3D, there are four main combinations of quadrature schemes, (GL, GL, GL), (TS, GL, GL), (GL, TS, GL), and (TS, TS, GL). Experiments indicate that the first two combinations yield similar results as each other, and likewise for the last two. As such, for brevity, the presented results are of (GL, GL, GL) and (TS, TS, GL), and are shown in Fig. 8. Similar observations as the two-dimensional case can be made:

---

[17] Quartile metrics are better suited in this analysis than, say, the mean and variance; for example, the exponential slope of the third quartile is unaffected when the first quartile errors begin to plateau around machine precision.

- For the randomly generated geometry of class $\mathcal{A}$, the quadrature accuracy using (GL, GL, GL) is generally superior to (TS, TS, GL). From the perspective of the first eliminated coordinate axis, the interfacial geometry of polynomials in $\mathcal{A}$ can be recast as the graph of a smooth height function devoid of branches, and as such we intuitively expect Gaussian quadrature to be effective.
- Similar to the 2D case, for polynomials in class $\mathcal{B}$, (GL, GL, GL) performs very poorly, which is to be expected given that every coordinate direction yields a height function with (near) vertical tangents, i.e., endpoint singularities. In this setting, tanh-sinh quadrature is better equipped at handling arbitrary endpoint singularities, and can be seen to ultimately attain higher convergence rates. Though, similar to the 2D case, Gaussian quadrature often yields better accuracy when the number of quadrature points is small; in this case, the transition point is around $q \approx 20$.
- One may also observe the wider spread in convergence rates, particularly of class $\mathcal{B}$ polynomials, compared to the 2D case. This is reflective of the increased interfacial complexity in 3D (as all examples in Fig. 6 show), as well as the more difficult geometries obtained in the dimension-reduced base integrals. We note here that if the randomly generated geometry is made smoother, by increasing the exponent α in the damping of higher-order Legendre modes (see (10)), then the convergence rates in both classes increase and the interquartile spread decreases; indeed, for α sufficiently large the randomly generated polynomials are almost linear and the implicitly defined domains essentially become polyhedral.

4.3.3. Summary

The above results, making use of randomly generated interfacial geometry, suggest the following guidelines for automatically choosing the type of one-dimensional quadrature in Algorithms 1 and 2:

(i) If it can be proven (using masking operations) that the polynomial(s) involved are the graph of a (potentially multi-valued) height function devoid of branching points/vertical tangents, then use Gauss-Legendre quadrature in the base integral;

(ii) Otherwise, and if high precision is required, consider applying tanh-sinh quadrature;

(iii) However, if only modest accuracy is required or a relatively small quadrature order q is to be used, then apply Gauss-Legendre quadrature.

Situations (i) and (iii) are applicable to the examples in class $\mathcal{A}$ as well as, for example, the vast majority of scenarios arising in cut-cell finite element methods of fluid-interface dynamics, see, e.g., [47,49].

The results also show that the quadrature algorithm is effective at handling complex interfacial topology, such as multi-component domains and varying degrees of curvature, without having to resort to heuristic subdivision methods. Through a suitable combination of one-dimensional quadrature types, approximately exponential convergence is obtained, i.e., doubling q approximately doubles the number of correct digits. The rate of convergence is related to the degree of curvature, and in the majority of cases, reasonably fast slopes are attained. Slower rates of convergence can be obtained in 3D, e.g., the results of Fig. 8(right) exhibit a wider spread of convergence rates above the median slope; these are associated with a high degree of localised interface curvature, as discussed further in later results. In these more rare scenarios, a small amount of constraint cell subdivision (i.e., localised h-refinement) can be effective in restoring rapid convergence; see, e.g., the supplementary material which contains a numerical study demonstrating this. Constraint cell refinement opens up the possibility of using different height directions on different parts of the interfacial geometry. However, we recall that an overarching design principle of these quadrature schemes was to avoid a reliance on such methods.

4.4. Bilinear example

In the next example, we consider a simple bilinear polynomial to study the effect of localised, high degree curvature. Consider the polynomial $\phi(x, y) = (x - \frac{1}{2})(y - \frac{1}{2}) - \epsilon^2$, whose interface depends on ϵ as follows: for ϵ sufficiently large, Γ exhibits a smooth rounded corner; as $\epsilon \to 0^+$, a sharp corner of curvature $\mathcal{O}(\epsilon^{-1})$ develops; and for $\epsilon = 0$, Γ is a cross centred at $(0.5, 0.5)$. Similar to the ellipsoidal test problem, we consider three kinds of integral quantities,

$$I_\Omega := \int_{U \cap \{\phi > 0\}} f, \quad I_\Gamma := \int_{U \cap \{\phi = 0\}} f, \quad I_{\Gamma_{\mathbf{n}}} := \int_{U \cap \{\phi = 0\}} f\mathbf{n},$$

where $U := (0, 1)^d$ is fixed, and the integrand $f : \mathbb{R}^2 \to \mathbb{R}$ is given by $f(x) = \cos(\frac{1}{4}\|x\|_2^2)$. We consider three values of ϵ, specifically 0, 0.01 and 0.1, representing the degenerate case, high localised curvature, and smooth geometry, respectively. Fig. 9 illustrates the corresponding quadrature schemes for $q = 3$, while Fig. 10 contains the results of a q-refinement study for the three integral quantities, using two kinds of quadrature choices, (TS, GL) and (GL, GL) (here we use the notation as in §4.3). Several observations can be made:

- When $\epsilon = 0$, the quadrature algorithm correctly handles the singular interface geometry and produces an intuitive quadrature scheme: for the volumetric integral, a tensor-product scheme is produced for the upper-right and lower-left
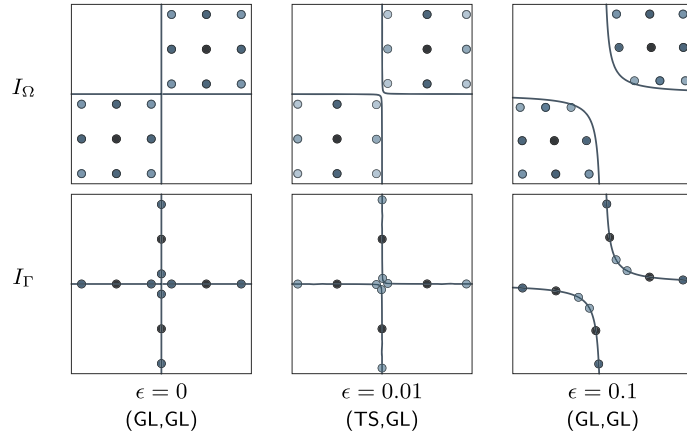
Fig. 9. Illustrations of the $q = 3$ quadrature schemes for the bilinear test problems considered in §4.4; in each instance, the quadrature points are shaded according to their weight, such that low-valued weights are pale and higher-valued weights are darker.
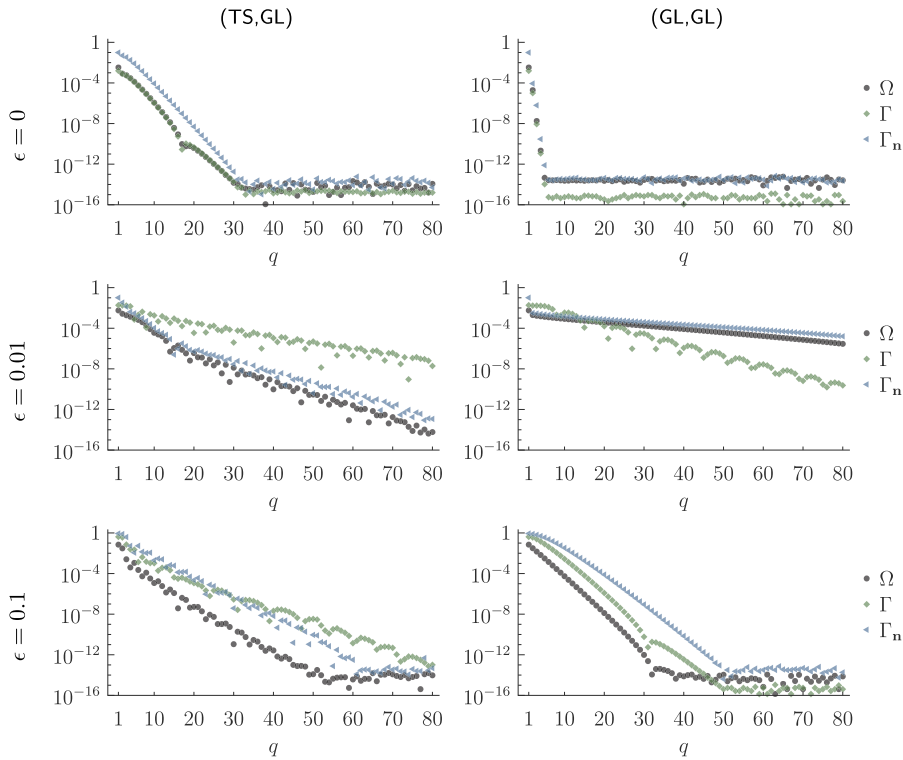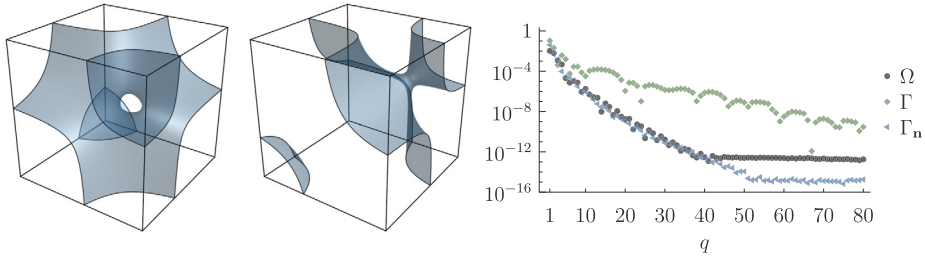

Fig. 10. Quadrature accuracy as a function of q for the bilinear test problems considered in §4.4. Here, • (black) denotes the relative error in the computed volumetric integral, I_Ω; ◆ (green) denotes the relative error in the computed surface integral in non-flux form, I_Γ; and ◀ (blue) denotes the relative error in the computed flux-form surface integral, $I_{\Gamma_\mathbf{n}}$.

    quadrants, and for the surface integral, the four arms of Γ are sampled with a simple one-dimensional scheme; see Fig. 9, left.

- For the relatively smooth interface corresponding to $\epsilon = 0.1$, the fastest convergence accuracy is obtained when Gauss-Legendre quadrature is used in all directions, as intuitively expected.

- In the high-curvature case of $\epsilon = 0.01$, exhibiting a rounded corner of radius about 1% of the cell width, the convergence rates of both schemes slow. In essence, there is a thin transition layer of width $\mathcal{O}(\epsilon)$ in the behaviour of the associated height functions. In the case of Gaussian quadrature, i.e., $(\mathsf{GL}, \mathsf{GL})$, this severely impacts its convergence rate. However, tanh-sinh quadrature is well-equipped at handling almost arbitrary endpoint singularities, helping to explain why the convergence rates of the $(\mathsf{TS}, \mathsf{GL})$ scheme are not as severely impacted.

Fig. 11. *(left pair)* Two different views of the interfacial geometry defined by the trilinear polynomial of §4.5; note the tunnel connecting one vertex of the unit cube $(0, 1)^3$ to the interior. *(right)* Quadrature accuracy as a function of q for the trilinear tunnel problem considered in §4.5. Here, • (black) denotes the relative error in the computed volumetric integral; ◆ (green) denotes the relative error in the computed surface integral in non-flux form; and ◀ (blue) denotes the relative error in the computed flux-form surface integral.
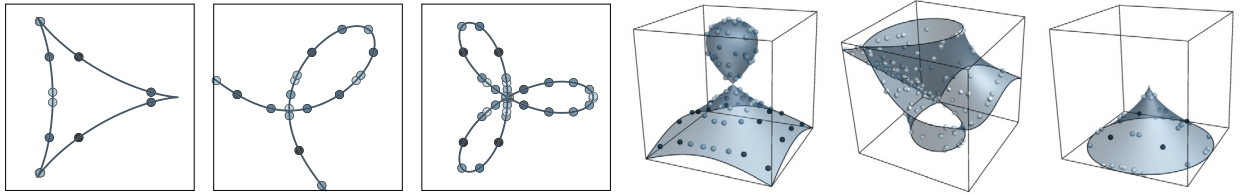
Fig. 12. Examples of the singularity test problems included in the supplementary material.

- In the case of $\epsilon = 0.01$ and the (TS, GL) scheme, note also the transition between a steep slope (mainly of the volumetric and surface-flux integrals) prior to $q \approx 15$ to a shallower slope after $q \approx 15$. This transition relates to the convergence of the interfacial geometry to the degenerate cross shape as $\epsilon \to 0$. For smaller and smaller ϵ, more and more quadrature points are needed to detect the diminishingly-small rounded corners of the interface; on the other hand, the integral error incurred by ignoring this geometry also diminishes as $\epsilon \to 0$. As such, the transition point ($q \approx 15$ for $\epsilon = 0.01$) will move to larger values of q as ϵ is made smaller, restoring the rapid convergence and effectively obtaining the results shown in Fig. 10, top row, once ϵ is sufficiently small.

Combined, these observations indicate that an appropriate choice of tanh-sinh and Gauss-Legendre quadrature can robustly handle a wide degree interface curvature. The above analysis considered these effects in the case the quadrature algorithm is "blind" to the interfacial geometry at hand. Naturally, if the geometry is known ahead of time, then one can subdivide appropriately so as to apply adaptive quadrature methods to achieve higher rates of convergence. Further comments about potential automated strategies for doing so are given in §5.

4.5. Trilinear tunnel

The next example demonstrates that even a low degree polynomial can yield non-trivial interfacial topology. Specifically, consider the trilinear polynomial $\phi : \mathbb{R}^3 \to \mathbb{R}$ given by

$$\phi(x, y, z) = 0.5 - 1.4z + 2.9xy - 6.5xyz + 3.2xz - 1.2x + 3.3yz - 1.3y.$$

The interface of ϕ in the unit cube $(0, 1)^3$ has two components, one of which exhibits a tunnel, see Fig. 11. Applied to the integrand $f(x) = \cos(\frac{1}{4}\|x\|_2^2)$, Fig. 11 also illustrates the results of a q-refinement study similar to those in prior studies, i.e., of the volumetric integral, surface integral in non-flux form, and flux-form surface integral of f. In particular, following the guidelines developed in §4.3, the computed results apply (TS, TS, GL) quadrature, i.e., Gauss-Legendre on the first eliminated axis, and tanh-sinh quadrature on every base integral. The results show similar convergence rates as in prior examples—once again, we observe that surface integrals computed in flux-form can be far more accurate than surface integrals in non-flux form.

4.6. Surfaces with singularities

The supplementary material accompanying this paper provides additional examples involving singularities, such as self-intersections and cusps; some are shown in Fig. 12. These examples are more contrived in the sense that they typically would not occur in computational physics problems, say; nonetheless, they serve to stress-test the quadrature algorithm in situations involving roots of high-order multiplicity.
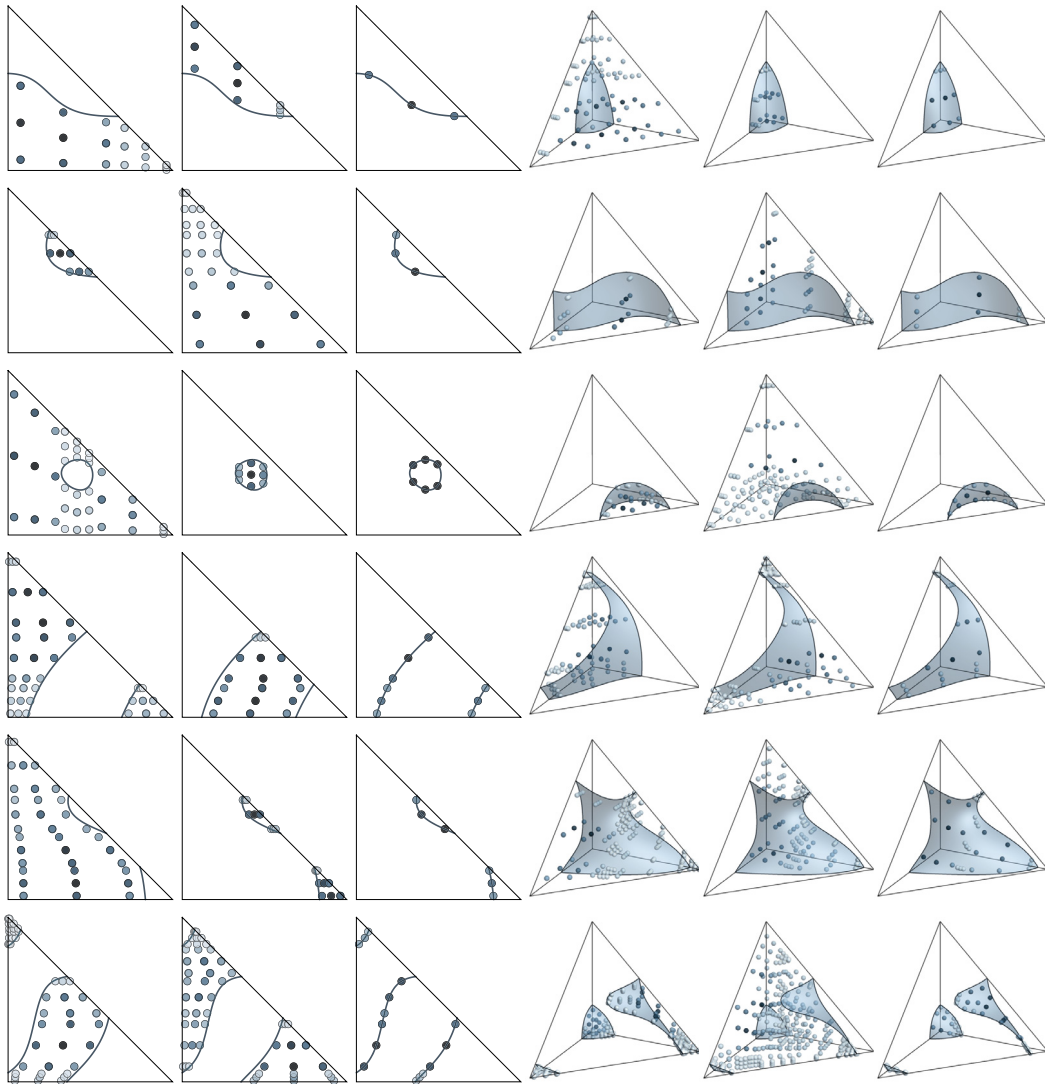
Fig. 13. Examples illustrating the adaptation of the quadrature algorithm to simplex constraint cells, following the mechanism discussed in §4.7.1. In each group of three, the left and middle figure corresponds to volumetric integrals over the implicitly defined regions $V \cap \{\phi < 0\}$ and $V \cap \{\phi > 0\}$, whereas the right figure corresponds to the surface integral over $V \cap \{\phi = 0\}$, where V is the standard triangle or tetrahedron simplex; in each instance, the quadrature points are shaded according to their weight, such that low-valued weights are pale and higher-valued weights are darker.

4.7. Application to simplices and intersecting geometry

By design, the quadrature algorithm can take in as input multiple multivariate polynomials. In this scenario, the implicitly defined interface is the union of each polynomial's zero isosurface, and the output of the algorithm is a quadrature scheme that can be used to compute integrals on individual connected components of the corresponding implicitly defined domain. We demonstrate this utility here, through some examples involving simplices and multi-component domains of intersecting geometry.

4.7.1. Extension to simplices

The dimension reduction process operates most naturally when the constraint cell $U \subset \mathbb{R}^d$ is a hyperrectangle. To handle the case of a simplex constraint cell, a simple approach is to extend the simplex to its enclosing hyperrectangle and then incorporate an additional polynomial $\psi : \mathbb{R}^d \to \mathbb{R}$ whose purpose is to cut the implicitly defined geometry along the sloping hyperplane of the target simplex. For example, suppose we wish to compute a quadrature scheme for

$$\int_{V \setminus \{\phi = 0\}} f,$$
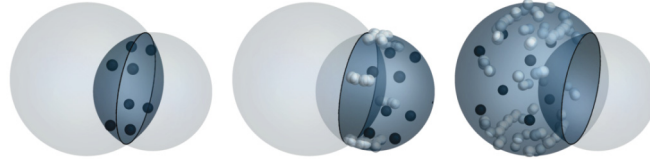
Fig. 14. An example showing how the quadrature algorithm can be used to build quadrature schemes for boolean operations on implicitly defined geometry, in this case of two overlapping spheres. In each figure, a volumetric quadrature is shown (using $q = 2$), for the lens-shaped intersection of both balls, and the two crescent-shaped regions (where one ball is subtracted from the other); the quadrature points are shaded according to their weight, such that low-valued weights are pale and higher-valued weights are darker.
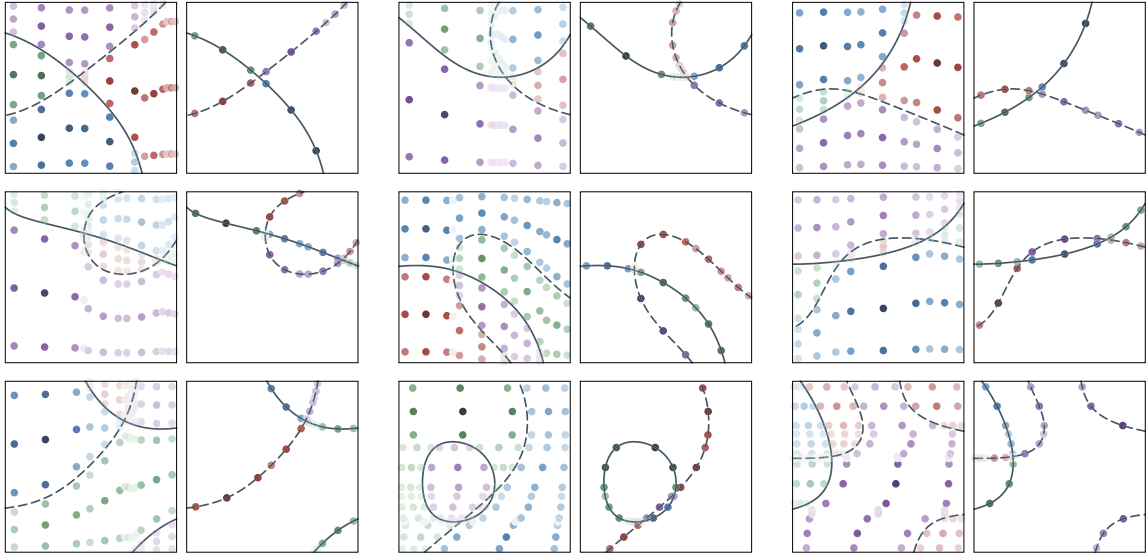
Fig. 15. Examples illustrating how the quadrature algorithm can be used to build quadrature schemes of intersecting geometry. In each pair, the left figure corresponds to volumetric integrals while the right figure corresponds to surface integrals. In each instance, the quadrature points are coloured according to the the signs of the input polynomials; each cluster of identical-coloured quadrature points yields a high-order quadrature scheme of the corresponding region.

where $V := \{x \in \mathbb{R}^d : x_i \geq 0 \text{ and } \sum_i x_i \leq 1\}$ is the standard unit simplex and ϕ is a polynomial. Define the linear polynomial ψ by $\psi(x) = -1 + \sum_i x_i$, and execute[18] Algorithm 1 to compute a volumetric quadrature scheme for the domain $U \setminus (\{\phi = 0\} \cup \{\psi = 0\})$, where $U = (0, 1)^d$. This yields a volumetric quadrature scheme filling all of U; one may then simply discard any quadrature nodes outside the simplex and leave the quadrature weights of all remaining nodes unmodified. This construction yields a high-order quadrature scheme appropriate for computing integrals over the connected components of $V \cap \{\phi \neq 0\}$. Examples are shown in Fig. 13. We note that Algorithms 1 and 2 could easily be modified to directly incorporate ψ for increased efficiency, e.g., so that the quadrature points in $U \setminus V$ are not needlessly computed.

4.7.2. Intersecting geometry

Finally, we show how the quadrature algorithm can be used to compute quadrature schemes for intersecting or overlapping domains. A general purpose mechanism is to: (a) invoke Algorithm 1 on the set of polynomials implicitly defining each domain, and then: (b) cluster the resulting quadrature points according to the signs of the input polynomials. This mechanism can be used to compute quadrature schemes for the various boolean operations, AND, OR, XOR, etc. By construction, each cluster yields a high-order accurate quadrature scheme when applied to sufficiently smooth integrands, yielding similar kinds of h-refinement and q-refinement convergence as shown in the other numerical experiments of this work. To illustrate the kind of schemes produces, Fig. 14 demonstrates an example involving two overlapping spheres and shows the volumetric quadrature schemes for the crescent- and lens-shaped regions; a similar clustering approach can be used to compute surface quadrature rules for these regions. Additional examples are shown in Fig. 15 of intersecting geometry implicitly defined by randomly generated 2D polynomials.[19]

---

[18] It is also worthwhile to modify the masks of all input polynomials such that the masks are disabled, i.e., value 0, in all subcells outside V; doing so helps to eliminate unneeded zeros of $\mathcal{P}$, $\mathcal{Q}$, or $\mathcal{R}$, i.e., helps to ignore the irrelevant branching or intersection points of the multi-valued height function which are outside the simplex.

[19] The same experiment can be done in 3D but the resulting geometry can be highly intricate, the visualisation of which is better suited for interactive plots rather than what can be shown on paper.

5. Concluding remarks

The quadrature algorithms developed in this work reinterpret the implicitly defined geometry as the graph of an implicitly defined, multi-valued height function. Using the corresponding splitting of coordinates, the target integral is reformulated as a base integral, in one fewer dimensions, whose associated integrand $\mathcal{F}$ is either a vertical line integral in the height direction (in the case of volumetric integrals), or a point-wise interfacial evaluation of the user-supplied integrand, taking into account curved surface Jacobian factors (in the case of surface integrals). Continuing recursively, the dimension reduction process eliminates one coordinate axis at a time, down to a one-dimensional base case; the base case then performs a one-dimensional quadrature, which in turn initiates a recursion back up the tree, with each new integrand evaluation performing a one-dimensional quadrature. To achieve a high order of accuracy, it is important to appropriately identify and treat the piecewise smooth behaviour of these one-dimensional integrals. This can be achieved through two mechanisms: (i) by dividing the domain of the base integral into one or more pieces, each one corresponding to the different characteristics of the multi-valued height function (e.g., crossing and branching points); and (ii) by appropriately choosing a quadrature scheme for the one-dimensional integrals (e.g., tanh-sinh quadrature to effectively handle endpoint singularities associated with branching points). In essence, the output of the algorithm is a quadrature scheme consisting of one or more patches, with each patch containing a tensor-product quadrature scheme warping to the implicitly defined geometry.[20]

Key findings include the following:

- Under the action of h-refinement and with q fixed, for sufficiently smooth geometry and integrands (e.g., away from singularities such as cusps), the order of accuracy of the quadrature scheme is approximately $2q$.
- Under the action of q-refinement, i.e., keeping both the constraint cell U and input polynomials $\{\phi_i\}$ fixed, the convergence is (asymptotically) approximately exponential in q, i.e., doubling q approximately doubles the number of accurate digits. In particular, the convergence behaviour and rate depends on the degree of curvature in the interface: smooth interfacial geometry yields rapid convergence (similar to Gaussian quadrature), whereas localised regions of very high curvature (e.g., nearly-sharp corners) can slow convergence; see, e.g., §4.4.
- Surface quadrature schemes in flux form, i.e., to compute $\int_\Gamma f\mathbf{n}$, are generally more accurate than those in non-flux form, i.e., $\int_\Gamma f$, relating to the property that the additional factor of $\mathbf{n}$ can remove (or cancel entirely) the behaviour of curved surface Jacobians. Fortunately, in many applications of surface quadrature, e.g., discontinuous Galerkin methods, it is more important to achieve high levels of accuracy in flux form than it is in non-flux form.
- In more contrived examples involving interfacial singularities, especially cusps, the numerical conditioning of the quadrature problem worsens, and this can impact the maximal possible accuracy in finite precision arithmetic. For example, the results of §4.6 show that some integrals can be limited to a maximum accuracy of $\mathcal{O}(\epsilon^\alpha)$, where ϵ is machine epsilon and $\alpha < 1$. This limited precision is related to the conditioning of computing roots near the singularities; at the singularities themselves, the roots are typically of high multiplicity. Here, two aspects are worth recalling: from a well-posedness standpoint, such problems are highly sensitive to perturbations in the input polynomials, e.g., small perturbations in a cusp can result in large changes in its surface area. On the other hand, these aspects of ill-conditioning do not always impact integral accuracy, e.g., full machine precision can be obtained for problems involving corners and junctions arising from crossing interfaces.

In this work we have mainly explored the use of Gauss-Legendre and tanh-sinh quadrature. In general, Gauss-Legendre performs well when the associated height functions do not have a vertical tangent, and, even if they do, it often yields a comparatively high accuracy when q is small. On the other hand, tanh-sinh is better equipped at handling end-point singularities, and is therefore more effective for height functions exhibiting vertical tangents, such as at branching points. Numerical experiments using randomly generated interfacial geometry, see §4.3, support this assessment and suggest the following strategy for choosing between Gauss-Legendre and tanh-sinh: if masking operations indicate there are branching points (i.e., the set $\mathcal{P}$ discussed in §3 is likely non-empty), and high precision is sought, then apply tanh-sinh quadrature in the base integral; otherwise, or if the target application is limited to a small q, use Gauss-Legendre quadrature.

From the outset, the quadrature algorithms were designed so as to not rely on ad hoc subdivision methods. Nevertheless, in some applications it could be highly beneficial to subdivide the constraint cell U into smaller subcells (of the same shape).

---

[20] We note here that the dimension reduction process is in some ways similar to the algorithm pioneered by Collins [12] for computing the *cylindrical algebraic decomposition* (CAD), a well-known tool in the field of real algebraic geometry [10,4]. Roughly speaking, a CAD is a decomposition of $\mathbb{R}^d$ into cells, such that a given set of polynomials have constant sign throughout each cell. A typical algorithm for computing a CAD operates through quantifier elimination and a projection phase (similar to finding a height direction and computing resultants in the description of the base integral) down to a one-dimensional base case, combined with a lifting phase which recursively adds dimensions to finish with a decomposition of the full space $\mathbb{R}^d$. Moreover, under certain conditions, each cell of a CAD is diffeomorphic to an open hypercube $(0, 1)^\ell$, for some $\ell \le d$. The patches mentioned above may be viewed as the image, under the action of the diffeomorphism, of a canonical tensor-product scheme in $(0, 1)^\ell$, suitably modified to account for the (implicitly defined) Jacobian determinant of this mapping. We also note that, in the present setting, we are considering implicitly defined geometry in a user-defined constraint cell U; a CAD could do the same if its input polynomials included a description of the boundary of U. For example, in 2D, if $U = (x_1, x_2) \times (y_1, y_2)$, then append the polynomials $x - x_1$, $x - x_2$, $y - y_1$, and $y - y_2$. In a sense, the quadrature algorithm directly incorporates this geometry, rather than have it be specified through additional polynomials. Finally, we note that the quadrature algorithm restricts attention to U, thus removing the computational expense of computing a full CAD, while the use of masking operations helps to eliminate additional CAD cells outside U.

Table 1

Timing measurements, in microseconds, of the build phase (i.e., execution of Algorithm 1), and construction phase (i.e., execution of Algorithm 2) for the indicated studies; see the main text for detail, §5. Experimental results obtained on an Intel Xeon E3-1535m v6 laptop, single core, operating at approximately 4 GHz.

Test problem	Build	Order q				
		1	2	4	8	16
2D ellipse, µs per grid cell with $U \cap \Gamma \neq \emptyset$	0.2	0.1	0.2	0.5	1.0	2.5
3D ellipsoid, µs per grid cell with $U \cap \Gamma \neq \emptyset$	1.5	0.2	0.8	2.2	8.8	50
2D randomly generated, average µs per instance	0.4	0.2	0.3	0.6	1.2	2.9
3D randomly generated, average µs per instance	6	0.9	2.2	8.0	25	120

Doing so effectively allows the choice of height direction/elimination axis to become more locally adaptive to the interfacial geometry at hand, such that different portions of the geometry use different height directions. For example, in the vast majority of cases of the randomly generated geometry in §4.3, just one or two levels of subdividing U into equal sized quadrants/octants, say, would yield the kinds of geometry for which Gauss-Legendre yields rapid convergence, faster than what is possible without any subdivision; see, e.g., the numerical results provided in the supplementary material. Of course, subdivision may incur a greater number of quadrature points. However, it should be noted that one cannot always rely on this kind of subdivision in order to resolve complex geometry of Γ; for example, singularities require a direct treatment and cannot be efficiently handled solely through constraint cell subdivision.

We have yet to discuss the computational cost of the quadrature algorithm; some comments and timing experiments are provided here. In short, the quadrature algorithm is fast for input polynomials of modest degree and can significantly outperform the algorithm developed in the author's prior work [51,50]. To provide an indication of the computational cost, Table 1 measures the time, in microseconds, needed to construct and evaluate the quadrature scheme for different values of q. In particular, the "Build" column measures the time needed to execute Algorithm 1 (computing masks, face restrictions, resultants, pseudo-discriminants, etc.) whereas the other columns indicate the time required to evaluate Algorithm 2 (evaluating vertical line restrictions of multivariate Bernstein polynomials, computing roots, mask filtering, sorting into subintervals, etc.), assuming that the cost of evaluating the inner-most user-supplied integrand is negligible. The first two rows correspond to the h-refinement study of the ellipsoid in §4.1, and measures the time per grid cell, in microseconds, needed to compute volumetric quadrature schemes, averaged over all grid cells containing the interface; meanwhile, the last two rows correspond to the randomly generated examples in §4.3 and measures the time per instance, in microseconds, needed to compute volumetric quadrature schemes, averaged over all randomly generated instances. Although the number of quadrature points scales as $\mathcal{O}(q^d)$, the timings in Table 1 do not directly follow this trend. This is because the quadrature formation is spread across a variety of components, such as calculating polynomial roots of mixed degree polynomials (sometimes using a fast hybrid of Bernstein subdivision plus Newton's method, sometimes using more complex eigenvalue methods), mask filtering operations, root sorting, and formation of the Gauss-Legendre/tanh-sinh quadrature points, all of which have different asymptotic complexities. Generally speaking, profiling experiments indicate that (at least for the test problems under consideration) polynomial root finding is not a bottleneck; instead, most of the time is spread between multivariate Bernstein polynomial manipulation, masking operations, and formation of the tensor-product scheme. We also note that our C++ implementation is not particularly optimised; nevertheless, the timing measurements reported in Table 1 are at least as fast as the identical ellipsoidal test carried out in [51], and in some cases more than three times faster.

We have not carefully considered here the issue of exploding polynomial degrees arising from the use of resultant and discriminant methods. In particular, the resultant of two tensor-product polynomials of degree $(p, \ldots, p) \in \mathbb{N}^d$ yields a tensor-product polynomial of maximal degree $(2p^2, \ldots, 2p^2) \in \mathbb{N}^{d-1}$ (see §3.4.2). Therefore, in a worst case scenario, a 2D quadrature problem, with input two polynomials of degree (p, p), may result in a degree $2p^2$ polynomial implicitly defining the domain of its base integral. In 3D, this effect compounds: a quadrature problem with input two polynomials of degree (p, p, p) may result in a 1D base integral (after elimination of two coordinate axes) involving a polynomial of degree $2(2p^2)^2 = 8p^4$. Meanwhile, the computational cost of building these polynomials has asymptotic complexity at least the degree squared, and so the worst-case complexity of the quadrature algorithm is at least $\mathcal{O}(p^4)$ in 2D and $\mathcal{O}(p^8)$ in 3D. (The preceding analysis ignores some aspects, such as perhaps needing eigenvalue methods to compute the roots of these polynomials, potentially with higher complexity.) Evidently, this cost, as well as associated aspects of numerical conditioning, may limit the practicality of the dimension reduction algorithm when the input polynomials describing the implicitly defined geometry are of high degree. Consequently, the quadrature algorithm developed in this work is perhaps best suited to applications involving low-to-medium degree multivariate polynomials. In any case, numerical experiments indicate that the algorithm is quite robust with respect to manipulating high degree polynomials, owing in part to the use of the Bernstein basis; see, e.g., some of the test problems included in the supplementary material.

Finally, we mention here some possibilities of further research that could lead to more advanced implementations of these quadrature algorithms:

- Several promising approaches could be implemented to increase quadrature adaptivity. In this work, and mainly for simplicity, we used a fixed number of quadrature points q per one-dimensional subinterval (see §3.6). Clearly, this could lead to an unnecessary number of quadrature points in the case of small subintervals. Thus, one could develop

more adaptive methods that expend fewer points in small subintervals, leading to a greater control in bounding the total number of output quadrature points. Extending this idea further, a more advanced implementation could begin with a low-order scheme, and adaptively add quadrature points where needed based on an error indicator, in the spirit of methods like Gauss-Kronrod, or through the re-use of existing points as afforded by, e.g., the nesting already present in tanh-sinh schemes. Guiding these efforts, one could use estimates of radius of convergence of the analyticity of the implicitly defined height functions, as quantified by the implicit function theorem. All of these latter ideas can exploit the fact the setup phase of the algorithm, i.e., Algorithm 1, is independent of the scheme evaluation phase, i.e., Algorithm 2.

- We explored here the use of tanh-sinh quadrature, owing to its effectiveness in treating almost arbitrary endpoint singularities. However, the implicitly defined height functions, here associated with domains implicitly defined by multivariate polynomials, have additional structure that could be exploited. For example, the endpoint singularities are typically associated with integrands which algebraically decay to zero. As such, alternatives to tanh-sinh quadrature which specifically target the behaviour of multi-valued height functions may yield higher rates of convergence.[21]

- The advances made with the above ideas could also open up other kinds of strategies for choosing the height direction/elimination axis, beyond the simple (though often effective) methods outlined in §3.5.

- A relatively straightforward method has been used to compute resultant and pseudo-discriminant polynomials (see §3.4), whose roots are then computed for use in the base integral. It may be possible to avoid the explicit construction of these polynomials (which can be become more ill-conditioned in high degree cases) and instead directly compute the roots through polynomial eigenvalue problems, see, e.g., [39,40].

- Finally, we mention the use of quadrature compression methods, see, e.g., [60]. These methods can be quite effective at significantly reducing the number of quadrature nodes of a pre-computed scheme while retaining its accuracy. However, the cost of compression can be comparatively high, hence these methods are best suited to applications in which this additional cost can be amortised.

CRediT authorship contribution statement

Robert I. Saye: Conceptualisation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Validation, Visualisation, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

Appendix A. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.jcp.2021.110720.

References

[1] R.W. Anderson, V.A. Dobrev, T.V. Kolev, R.N. Rieben, V.Z. Tomov, High-order multi-material ALE hydrodynamics, SIAM J. Sci. Comput. 40 (2018) B32–B58, https://doi.org/10.1137/17M1116453.

[2] D.H. Bailey, Tanh-sinh high-precision quadrature, https://www.davidhbailey.com/dhbpapers/dhb-tanh-sinh.pdf, 2006. (Accessed 2 April 2021) (online).

[3] D.H. Bailey, Y. Hida, X.S. Li, B. Thompson, K. Jeyabalan, A. Kaiser, QD library, https://www.davidhbailey.com/dhbsoftware/, 2014.

[4] S. Basu, R. Pollack, M.-F. Roy, Algorithms in Real Algebraic Geometry, 2 ed., Springer-Verlag, Berlin, Heidelberg, 2006.

[5] M. Beśka, Convexity and variation diminishing property of multidimensional Bernstein polynomials, Approx. Theory Appl. 5 (1989) 59–78, https://doi.org/10.1007/BF02888887.

[6] D. Bini, L. Gemignani, Bernstein–Bezoutian matrices, in: Algebraic and Numerical Algorithms, Theor. Comput. Sci. 315 (2004) 319–333, https://doi.org/10.1016/j.tcs.2004.01.016.

[7] D. Bochkov, F. Gibou, Solving Poisson-type equations with Robin boundary conditions on piecewise smooth interfaces, J. Comput. Phys. 376 (2019) 1156–1198, https://doi.org/10.1016/j.jcp.2018.10.020.

---

[21] Gauss-Jacobi is one possibility, but this approach requires precise determination of the algebraic power of the end-point singularity, which could vary from segment to segment of the height function, thus making the approach intricate to implement.

[8] J.U. Brackbill, D.B. Kothe, C. Zemach, A continuum method for modeling surface tension, J. Comput. Phys. 100 (1992) 335–354, https://doi.org/10.1016/0021-9991(92)90240-Y.

[9] E. Burman, S. Claus, P. Hansbo, M.G. Larson, A. Massing, CutFEM: discretizing geometry and partial differential equations, Int. J. Numer. Methods Eng. 104 (2015) 472–501, https://doi.org/10.1002/nme.4823.

[10] B.F. Caviness, J.R. Johnson (Eds.), Quantifier Elimination and Cylindrical Algebraic Decomposition, Texts & Monographs in Symbolic Computation, Springer-Verlag, Wien, 1998.

[11] K.W. Cheng, T.-P. Fries, Higher-order XFEM for curved strong and weak discontinuities, Int. J. Numer. Methods Eng. 82 (2010) 564–590, https://doi.org/10.1002/nme.2768.

[12] G.E. Collins, Quantifier elimination for real closed fields by cylindrical algebraic decomposition, in: H. Brakhage (Ed.), Automata Theory and Formal Languages, Springer Berlin Heidelberg, Berlin, Heidelberg, 1975, pp. 134–183.

[13] T. Cui, W. Leng, H. Liu, L. Zhang, W. Zheng, High-order numerical quadratures in a tetrahedron with an implicitly defined curved interface, ACM Trans. Math. Softw. 46 (2020), https://doi.org/10.1145/3372144.

[14] M. Discacciati, A. Quarteroni, S. Quinodoz, Numerical approximation of internal discontinuity interface problems, SIAM J. Sci. Comput. 35 (2013) A2341–A2369, https://doi.org/10.1137/110850487.

[15] L. Drescher, H. Heumann, K. Schmidt, A high order method for the approximation of integrals over implicitly defined hypersurfaces, SIAM J. Numer. Anal. 55 (2017) 2592–2615, https://doi.org/10.1137/16M1102227.

[16] B. Engquist, A.-K. Tornberg, R. Tsai, Discretization of Dirac delta functions in level set methods, J. Comput. Phys. 207 (2005) 28–51, https://doi.org/10.1016/j.jcp.2004.09.018.

[17] L. Engvall, J.A. Evans, Isogeometric unstructured tetrahedral and mixed-element Bernstein–Bézier discretizations, Comput. Methods Appl. Mech. Eng. 319 (2017) 83–123, https://doi.org/10.1016/j.cma.2017.02.017.

[18] R. Farouki, V. Rajan, Algorithms for polynomials in Bernstein form, Comput. Aided Geom. Des. 5 (1988) 1–26, https://doi.org/10.1016/0167-8396(88)90016-7.

[19] R.T. Farouki, T.N.T. Goodman, On the optimal stability of the Bernstein basis, Math. Comput. 65 (1996) 1553–1566, https://doi.org/10.1090/S0025-5718-96-00759-4.

[20] T. Fries, S. Omerović, D. Schöllhammer, J. Steidl, Higher-order meshing of implicit geometries—part I: integration and interpolation in cut elements, Comput. Methods Appl. Mech. Eng. 313 (2017) 759–784, https://doi.org/10.1016/j.cma.2016.10.019.

[21] T.-P. Fries, S. Omerović, Higher-order accurate integration of implicit geometries, Int. J. Numer. Methods Eng. 106 (2016) 323–371, https://doi.org/10.1002/nme.5121.

[22] J. Garloff, The Bernstein algorithm, Interval Comput. 2 (1993) 154–168.

[23] D. Gunderman, K. Weiss, J.A. Evans, High-accuracy mesh-free quadrature for trimmed parametric surfaces and volumes, https://arxiv.org/abs/2101.06497, 2021.

[24] D. Gunderman, K. Weiss, J.A. Evans, Spectral mesh-free quadrature for planar regions bounded by rational parametric curves, Comput. Aided Des. 130 (2021) 102944, https://doi.org/10.1016/j.cad.2020.102944.

[25] N. Hale, A. Townsend, Fast and accurate computation of Gauss–Legendre and Gauss–Jacobi quadrature nodes and weights, SIAM J. Sci. Comput. 35 (2013) A652–A674, https://doi.org/10.1137/120889873.

[26] D. Henrion, J.B. Lasserre, C. Savorgnan, Approximate volume and integration for basic semialgebraic sets, SIAM Rev. 51 (2009) 722–743, https://doi.org/10.1137/080730287.

[27] D.J. Holdych, D.R. Noble, R.B. Secor, Quadrature rules for triangular and tetrahedral elements with generalized functions, Int. J. Numer. Methods Eng. 73 (2008) 1310–1327, https://doi.org/10.1002/nme.2123.

[28] J. Hrivnák, L. Motlochová, J. Patera, Cubature formulas of multivariate polynomials arising from symmetric orbit functions, Symmetry 8 (2016) 63, https://doi.org/10.3390/sym8070063.

[29] A. Johansson, B. Kehlet, M.G. Larson, A. Logg, Multimesh finite element methods: solving PDEs on multiple intersecting meshes, Comput. Methods Appl. Mech. Eng. 343 (2019) 672–689, https://doi.org/10.1016/j.cma.2018.09.009.

[30] A. Johansson, M.G. Larson, A high order discontinuous Galerkin Nitsche method for elliptic problems with fictitious boundary, Numer. Math. 123 (2013) 607–628, https://doi.org/10.1007/s00211-012-0497-1.

[31] C. Kublik, R. Tsai, An extrapolative approach to integration over hypersurfaces in the level set framework, Math. Comput. 87 (2018) 2365–2392, https://doi.org/10.1090/mcom/3282.

[32] P. Lairez, M. Mezzarobba, M. Safey El Din, Computing the volume of compact semi-algebraic sets, in: Proceedings of the 2019 on International Symposium on Symbolic and Algebraic Computation, ISSAC '19, 2019, pp. 259–266.

[33] C. Lehrenfeld, High order unfitted finite element methods on level set domains using isoparametric mappings, Comput. Methods Appl. Mech. Eng. 300 (2016) 716–733, https://doi.org/10.1016/j.cma.2015.12.005.

[34] Q. Lin, J. Rokne, Methods for bounding the range of a polynomial, J. Comput. Appl. Math. 58 (1995) 193–199, https://doi.org/10.1016/0377-0427(93)E0270-V.

[35] C. Min, F. Gibou, Geometric integration over irregular domains with application to level-set methods, J. Comput. Phys. 226 (2007) 1432–1443, https://doi.org/10.1016/j.jcp.2007.05.032.

[36] C. Min, F. Gibou, Robust second-order accurate discretizations of the multi-dimensional Heaviside and Dirac delta functions, J. Comput. Phys. 227 (2008) 9686–9695, https://doi.org/10.1016/j.jcp.2008.07.021.

[37] N. Moës, J. Dolbow, T. Belytschko, A finite element method for crack growth without remeshing, Int. J. Numer. Methods Eng. 46 (1999) 131–150, https://doi.org/10.1002/(SICI)1097-0207(19990910)46:1<131::AID-NME726>3.0.CO;2-J.

[38] B. Müller, F. Kummer, M. Oberlack, Highly accurate surface and volume integration on implicit domains by means of moment-fitting, Int. J. Numer. Methods Eng. 96 (2013) 512–528, https://doi.org/10.1002/nme.4569.

[39] Y. Nakatsukasa, V. Noferini, A. Townsend, Computing the common zeros of two bivariate functions via Bézout resultants, Numer. Math. 129 (2015) 181–209, https://doi.org/10.1007/s00211-014-0635-z.

[40] V. Noferini, A. Townsend, Numerical instability of resultant methods for multidimensional rootfinding, SIAM J. Numer. Anal. 54 (2016) 719–743, https://doi.org/10.1137/15M1022513.

[41] S. Osher, R. Fedkiw, Level Set Methods and Dynamic Implicit Surfaces, Applied Mathematical Sciences, Springer, 2003.

[42] S. Osher, J.A. Sethian, Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations, J. Comput. Phys. 79 (1988) 12–49, https://doi.org/10.1016/0021-9991(88)90002-2.

[43] S. Pan, X. Hu, N.A. Adams, A consistent analytical formulation for volume estimation of geometries enclosed by implicitly defined surfaces, SIAM J. Sci. Comput. 40 (2018) A1523–A1543, https://doi.org/10.1137/17M1126370.

[44] C.S. Peskin, The immersed boundary method, Acta Numer. 11 (2002) 479–517, https://doi.org/10.1017/S0962492902000077.

[45] J. Rajyaguru, M.E. Villanueva, B. Houska, B. Chachuat, Chebyshev model arithmetic for factorable functions, J. Glob. Optim. 68 (2017) 413–438, https://doi.org/10.1007/s10898-016-0474-9.

[46] J.A. Reeger, B. Fornberg, M.L. Watts, Numerical quadrature over smooth, closed surfaces, Proc. R. Soc. A, Math. Phys. Eng. Sci. 472 (2016) 20160401, https://doi.org/10.1098/rspa.2016.0401.

[47] R. Saye, Interfacial gauge methods for incompressible fluid dynamics, Sci. Adv. 2 (2016), https://doi.org/10.1126/sciadv.1501869.

[48] R. Saye, Implicit mesh discontinuous Galerkin methods and interfacial gauge methods for high-order accurate interface dynamics, with applications to surface tension dynamics, rigid body fluid-structure interaction, and free surface flow: part I, J. Comput. Phys. 344 (2017) 647–682, https://doi.org/10.1016/j.jcp.2017.04.076.

[49] R. Saye, Implicit mesh discontinuous Galerkin methods and interfacial gauge methods for high-order accurate interface dynamics, with applications to surface tension dynamics, rigid body fluid-structure interaction, and free surface flow: part II, J. Comput. Phys. 344 (2017) 683–723, https://doi.org/10.1016/j.jcp.2017.05.003.

[50] R. Saye, Algoim – algorithms for implicitly defined geometry, level set methods, and Voronoi implicit interface methods, https://algoim.github.io/, 2019. (Accessed 31 March 2021).

[51] R.I. Saye, High-order quadrature methods for implicitly defined surfaces and volumes in hyperrectangles, SIAM J. Sci. Comput. 37 (2015) A993–A1019, https://doi.org/10.1137/140966290.

[52] R.I. Saye, Fast multigrid solution of high-order accurate multiphase Stokes problems, Commun. Appl. Math. Comput. Sci. 15 (2020) 147–196, https://doi.org/10.2140/camcos.2020.15.33.

[53] F. Scholz, B. Jüttler, Numerical integration on trimmed three-dimensional domains with implicitly defined trimming surfaces, Comput. Methods Appl. Mech. Eng. 357 (2019) 112577, https://doi.org/10.1016/j.cma.2019.112577.

[54] F. Scholz, B. Jüttler, High-order quadrature on planar domains based on transport theorems for implicitly defined moving curves, Tech. Rep. 89, Nationales Forschungsnetzwerk, Geometry+ Simulation (May 2020), 2020.

[55] P. Schwartz, J. Percelay, T.J. Ligocki, H. Johansen, D.T. Graves, D. Devendran, P. Colella, E. Ateljevich, High-accuracy embedded boundary grid generation using the divergence theorem, Commun. Appl. Math. Comput. Sci. 10 (2015) 83–96, https://doi.org/10.2140/camcos.2015.10.83.

[56] J.A. Sethian, Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Sciences, Cambridge University Press, 1999.

[57] J.A. Sethian, P. Smereka, Level set methods for fluid interfaces, Annu. Rev. Fluid Mech. 35 (2003) 341–372, https://doi.org/10.1146/annurev.fluid.35.101101.161105.

[58] P. Smereka, The numerical approximation of a delta function with application to level set methods, J. Comput. Phys. 211 (2006) 77–90, https://doi.org/10.1016/j.jcp.2005.05.005.

[59] A. Sommariva, M. Vianello, Gauss–Green cubature and moment computation over arbitrary geometries, J. Comput. Appl. Math. 231 (2009) 886–896, https://doi.org/10.1016/j.cam.2009.05.014.

[60] Y. Sudhakar, A. Sommariva, M. Vianello, W.A. Wall, On the use of compressed polyhedral quadrature formulas in embedded interface methods, SIAM J. Sci. Comput. 39 (2017) B571–B587, https://doi.org/10.1137/16M1085206.

[61] H. Takahasi, M. Mori, Double exponential formulas for numerical integration, Publ. Res. Inst. Math. Sci. 9 (1974) 721–741, https://doi.org/10.2977/prims/1195192451.

[62] V. Thiagarajan, V. Shapiro, Adaptively weighted numerical integration over arbitrary domains, Comput. Math. Appl. 67 (2014) 1682–1702, https://doi.org/10.1016/j.camwa.2014.03.001.

[63] A.-K. Tornberg, B. Engquist, Numerical approximations of singular source terms in differential equations, J. Comput. Phys. 200 (2004) 462–488, https://doi.org/10.1016/j.jcp.2004.04.011.

[64] J.D. Towers, Two methods for discretizing a delta function supported on a level set, J. Comput. Phys. 220 (2007) 915–931, https://doi.org/10.1016/j.jcp.2006.05.037.

[65] L.N. Trefethen, J.A.C. Weideman, The exponentially convergent trapezoidal rule, SIAM Rev. 56 (2014) 385–458, https://doi.org/10.1137/130932132.

[66] C. Uchytil, D. Storti, A coarea formulation for grid-based evaluation of volume integrals, J. Comput. Inf. Sci. Eng. 20 (2020), https://doi.org/10.1115/1.4047355.

[67] J. Vanherck, B. Sorée, W. Magnus, Tanh-sinh quadrature for single and multiple integration using floating-point arithmetic, https://arxiv.org/abs/2007.15057, 2020.

[68] G. Ventura, On the elimination of quadrature subcells for discontinuous functions in the eXtended finite-element method, Int. J. Numer. Methods Eng. 66 (2006) 761–795, https://doi.org/10.1002/nme.1570.

[69] X. Wen, High order numerical quadratures to one dimensional delta function integrals, SIAM J. Sci. Comput. 30 (2008) 1825–1846, https://doi.org/10.1137/070682939.

[70] X. Wen, High order numerical methods to two dimensional delta function integrals in level set methods, J. Comput. Phys. 228 (2009) 4273–4290, https://doi.org/10.1016/j.jcp.2009.03.004.

[71] X. Wen, High order numerical methods to three dimensional delta function integrals in level set methods, SIAM J. Sci. Comput. 32 (2010) 1288–1309, https://doi.org/10.1137/090758295.

[72] J.R. Winkler, A resultant matrix for scaled Bernstein polynomials, Linear Algebra Appl. 319 (2000) 179–191, https://doi.org/10.1016/S0024-3795(00)00189-0.

[73] J.R. Winkler, A companion matrix resultant for Bernstein polynomials, Linear Algebra Appl. 362 (2003) 153–175, https://doi.org/10.1016/S0024-3795(02)00486-X.

[74] J.R. Winkler, A unified approach to resultant matrices for Bernstein basis polynomials, in: Solid and Physical Modeling, Comput. Aided Geom. Des. 25 (2008) 529–541, https://doi.org/10.1016/j.cagd.2007.09.004.

[75] J.R. Winkler, N. Yang, Resultant matrices and the computation of the degree of an approximate greatest common divisor of two inexact Bernstein basis polynomials, Comput. Aided Geom. Des. 30 (2013) 410–429, https://doi.org/10.1016/j.cagd.2013.01.003.

[76] M. Yurtoglu, M. Carton, D. Storti, Treat all integrals as volume integrals: a unified, parallel, grid-based method for evaluation of volume, surface, and path integrals on implicitly defined domains, J. Comput. Inf. Sci. Eng. 18 (2018), https://doi.org/10.1115/1.4039639.

[77] S. Zahedi, A.-K. Tornberg, Delta function approximations in level set methods by distance function extension, J. Comput. Phys. 229 (2010) 2199–2219, https://doi.org/10.1016/j.jcp.2009.11.030.