High-Order Continuous Geometrical Validity

FEDERICO SICHETTI, University of Genoa, Italy
ZIZHOU HUANG, New York University, United States
MARCO ATTENE, CNR-IMATI: Genova, Italy
DENIS ZORIN, New York University, United States
ENRICO PUPPO, University of Genoa, Italy
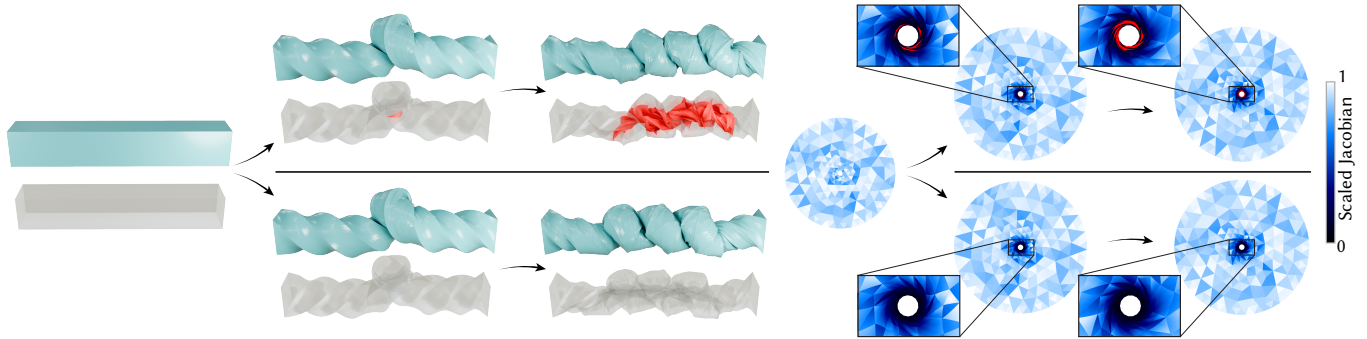DANIELE PANOZZO, New York University, United States

Fig. 1. Soft body simulations [Ferguson et al. 2023] of twisting a beam in 3D (left) and twisting the inner circle of a fat ring in 2D (right), without (top) and with (bottom) our continuous validity check. The high-order invalid elements are marked in red, and the scaled Jacobian distribution is shown on the right. Without our method, the Twist-beam example has 1 and 161 invalid elements at t=3 s and t=7 s respectively; the Twist-ring example has 11 and 31 invalid elements at t=0.4 s and t=1 s respectively. With our validity check and adaptive quadrature refinement scheme, the simulations do not contain any invalid elements.

We propose a conservative algorithm to test the geometrical validity of simplicial (triangles, tetrahedra), tensor product (quadrilaterals, hexahedra), and mixed (prisms) elements of arbitrary polynomial order as they deform linearly within a time interval.

Our algorithm uses a combination of adaptive Bézier refinement and bisection search to determine if, when, and where the Jacobian determinant of an element's polynomial geometric map becomes negative in the transition from one configuration to another. In elastodynamic simulation, our algorithm guarantees that the system remains physically valid during the entire trajectory, not only at discrete time steps. Unlike previous approaches, physical validity is preserved even when our method is implemented using floating point arithmetic. Hence, our algorithm is only slightly slower than existing non-conservative methods while providing guarantees and while being an easy drop-in replacement for current validity tests.

To prove the practical effectiveness of our algorithm, we demonstrate its use in a high-order Incremental Potential Contact (IPC) elastodynamic simulator and experimentally show that it prevents invalid, simulation-breaking configurations that would otherwise occur using non-conservative methods.

Authors' addresses: Federico Sichetti, University of Genoa, Italy, federico.sichetti@edu.unige.it; Zizhou Huang, New York University, United States, zizhou@nyu.edu; Marco Attene, CNR-IMATI: Genova, Italy, marco.attene@ge.imati.cnr.it; Denis Zorin, New York University, United States, dzorin@cs.nyu.edu; Enrico Puppo, University of Genoa, Italy, enrico.puppo@unige.it; Daniele Panozzo, New York University, United States, panozzo@nyu.edu.

1 INTRODUCTION

In computer graphics, mechanical engineering, and scientific computing, physical objects are often modeled using meshes composed of simple elements, such as tetrahedra, hexahedra, and prisms. Each element is typically associated with two maps: (1) a geometric map that defines the element's shape; and (2) a basis map that extends quantities (such as displacement or velocities) defined at the element's nodes into its interior. For rendering, linear polynomials (hat functions) are commonly used for both maps, but higher-order versions are widely employed when greater accuracy is required. When the basis map is used to interpolate a *displacement*, the element's geometry is derived by combining the initial geometric map with the basis map, yielding a polynomial whose order corresponds to the highest of the two. A typical example in graphics is the use of second-order elements in finite element (FE) simulations for fabrication [Panetta et al. 2015], which generates quadratic curved elements even if the initial geometric map is piecewise linear.

In the following, we will overload the term *geometry map* to describe the final geometry of an object, which can thus also incorporate the displacement basis map. In mathematical notation, this is defined with a polynomial function:

$$x : \sigma \longrightarrow \mathbb{R}^n,$$

where σ represents a reference element domain (such as a standard simplex or multi-interval), and n denotes the dimension of the element. See Figure 2 for examples.

During animation or simulation, objects are deformed by modifying the geometry map, the basis map, or both: these modifications are usually performed by applying linear transformations to the map coefficients. Note that, for a piecewise linear geometric map, this reduces to the usual interpolation of the coordinates of the mesh vertices. Similarly, for curved meshing, it is also typical to start with a piecewise linear mesh and then curve its elements to reduce the approximation error [Toulorge et al. 2013].

Since these meshes are used to represent physical objects, a basic, yet elusive, requirement is that the geometry of the object does not self-intersect at any time during deformation. This condition can be violated in two ways: (1) by a global intersection or (2) by a local lack of injectivity of the geometric map. The first problem has been studied thoroughly in the parametrization and simulation literature, at least for linear elements [Smith and Schaefer 2015; Wang et al. 2021a]. The second condition has been considered mostly in the context of mesh parametrization, and, even in that case, no robust algorithm exists (see Section 3). In this work, we focus on the latter.

Problem statement. We denote with $\bar{x}$ a geometric map that is dynamically changing over time and we study its injectivity. This reduces to studying the sign of the determinant of its Jacobian $|J_{\bar{x}}|$ to assess the continuous validity of the element, i.e., if, where and when $|J_{\bar{x}}|$ becomes negative. We note that this is a subtly challenging problem even for the simple case of a linear triangle (Figure 3): an element that is valid in its initial and final configurations might become invalid *during* the trajectory.

Conservative answer. An indispensable requirement for the robustness of simulations is that they employ *conservative* algorithms: that is, a program that only reports results whose correctness cannot be influenced by approximation errors. Thus, a Boolean predicate (such as asking whether an element is valid or not) is required to return a third "undecided" value if the approximation error is large enough to make the naive result unreliable. Conversely, if the algorithm returns an answer, that answer must be provably correct. In the context of continuous validity, we wish to find the maximum time step t^* such that the element is provably valid at all times in $[0, t^*]$: a conservative answer can be any value $t \in [0, t^*)$. Obviously, in practice, it is also desirable that t be as close to t^* as possible.

Brief summary of the state of the art. A common approach among practitioners is testing the validity of elements by just computing the value of $|J_x|$ at quadrature points at every time step, as done, e.g., by Dey et al. [2001]. This might fail to detect invalidity even in static tests, let alone the continuous case. Smith and Schaefer [2015] study the continuous problem for the specific case of 2D linear triangles, reducing it to a quadratic root finding, which is

solved numerically; this approach is hard to extend to 3D or to higher-order polynomials. Other approaches exist to resolve the static test for high-order elements [Johnen et al. 2014], but they are neither conservative nor easily extensible to the continuous test. More detail on the state of the art is provided in Section 3.

Contribution. We introduce the first generic formulation and algorithm for the continuous validity test of elements, supporting the most common types — such as triangles, quadrilaterals, tetrahedra, prisms, and hexahedra — and extending to high-order basis and geometric maps. Our algorithm is provably conservative when implemented using floating-point arithmetic, meaning that if an element is detected to be valid, it is guaranteed to remain valid throughout the entire specified time interval. This level of robustness, crucial for algorithmic reliability, has not been achieved by any previous method.

If an element becomes invalid at any point, our algorithm provides a conservative estimate of the inversion time and introduces a custom quadrature rule that accurately reflects the detected inversion. Specifically, this means that the numerical integration diverges when the element inverts, a property not provided by adaptive quadrature rules commonly used in high-order finite elements.

While designed for the dynamic case, our algorithm can also be used for the static case, with minor modifications: in this setting, our algorithm is the first algorithm to provide a conservative static geometrical validity test for high-order elements.

Evaluation. Our algorithm is designed and implemented for high performance, as its use-case is within optimization loops requiring the testing of large datasets: on static checks, we demonstrate that our test is competitive in terms of runtime with current non-conservative methods, being slightly slower while guaranteeing a conservative answer. To quantitatively evaluate the correctness and efficiency of our approach and compare it with more specialized alternatives, we construct a dataset of 2D and 3D time-dependent queries whose ground truth is computed using (extremely expensive) symbolic root finding.

Applications. Having access to a conservative check we discovered that it is very common for high-order FE simulations to contain invalid elements in their solution; we show examples in PolyFEM in Figures 1 and 12. This seems to be a common problem with high-order FE codes: for example, FEBio [Maas et al. 2012] also uses a static check only at quadrature points. This issue is rarely mentioned in the literature [Anderson et al. 2014; Dobrev et al. 2019] and we are not aware of other papers proposing a solution. We believe that the presence of invalidity is due to the use of insufficiently accurate quadrature to capture the infinite elastic potential inside some of the most distorted elements. This is a major source of both the numerical fragility of this software and inaccuracy in the solution as physically invalid configurations are reported as the simulation result. By replacing the validity check and the quadrature in PolyFEM with our approach, we show that these issues disappear and the impact on performance is moderate.

Impact. We believe our algorithm will be an essential addition to the growing toolkit of robust geometric building blocks used in modern parametrization, meshing, and simulation algorithms. To foster
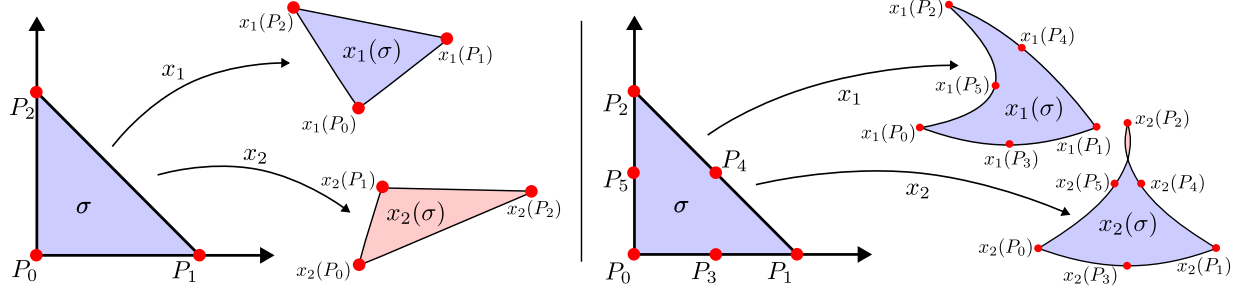
Fig. 2. A reference domain σ is mapped to linear elements (Left) and quadratic elements (Right). Quadratic elements can be given either by an initial quadratic geometry or by combining an initial linear geometry with a quadratic displacement. Geometric maps on a Lagrangian basis are specified by mapping the domain points P_i to control points $x_j(P_i)$. Blue and red areas denote a positive and negative determinant of the Jacobian, respectively. In both cases, element $x_2(\sigma)$ is invalid. In the linear case, the whole element is inverted, while in the quadratic case, only a small portion of $x_2(\sigma)$ is inverted.
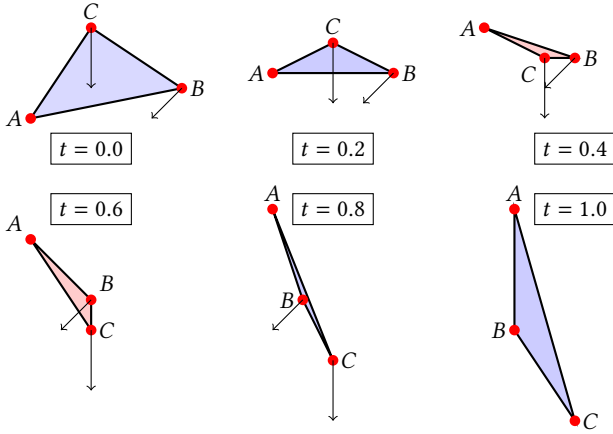

Fig. 3. A dynamic linear element with linear trajectories that flips twice in a time interval, resulting in an element that is valid at both time steps, but invalid in the transition. Assuming point A remains fixed, the arrows represent the velocities of points B and C to reach the final position. The color of each element indicates the sign of the determinant at time t: blue if positive and red if negative. The dynamic element's Jacobian determinant is a univariate polynomial in t of degree 2, with two distinct roots in $[0, 1]$.

its adoption, we provide an open-source reference implementation at https://gitlab.com/fsichetti/hocgv.

2 OVERVIEW OF THE METHOD

Given a dynamic element $\bar{x}(\sigma)$ deforming linearly over the time interval $[0, 1]$, we study the determinant of its Jacobian $|J_{\bar{x}}|$ (Sections 5 and 6). The polynomial $|J_{\bar{x}}|$ can have a high order in both its spatial and temporal variables (Appendix A). To ensure a conservative answer, we employ a custom bisection root-finding method controlled by an accuracy parameter $\delta > 0$. Let t^* denote the earliest time at which $\bar{x}(\sigma)$ becomes invalid (i.e., $|J_{\bar{x}}|$ turns negative). We return a time $\underline{t^*} > t^* - \delta$ and a point $\underline{P} \in \sigma$ such that $|J_{\bar{x}}|$ is positive everywhere for $t \leq \underline{t^*}$, and $|J_{\bar{x}}(\underline{P}, t)|$ becomes negative for some $t \leq \underline{t^*} + \delta$. This guarantees that the element can safely deform up to time $\underline{t^*}$ while the point $\underline{P}$ is used to adaptively refine the quadrature

rule, steering the simulation away from invalid configurations (Section 7). If the element remains valid throughout, we simply return $\underline{t^*} = 1$.

The parameter domain of a dynamic element has both space and time dimensions. Our test proceeds by bisecting the time dimension: it maintains lower $\underline{t^*}$ and upper $\overline{t^*}$ bounds for the critical inversion time, and terminates when the difference between these bounds is smaller than δ, or when the element is confirmed to be valid throughout the interval. The algorithm employs a priority queue of sub-domains, which are created by recursively splitting the initial space-time domain.

For a given sub-domain S, we compute a *minimum inclusion function* that returns an interval I, which is guaranteed to contain the minimum value of $|J_{\bar{x}}|(S)$: if I is strictly positive, the element is valid in S; if I is strictly negative, the element is invalid somewhere within S (but it is not necessarily invalid everywhere in S); if I contains zero, nothing can be said and further refinement of S is necessary. The minimum inclusion function is a crucial component of our method; details on its definition and computation are provided in Sections 4.3 and 6.1, respectively.

Another key aspect of the algorithm is the decoupling of refinements in the spatial and temporal dimensions. While the time dimension may require refinement until the interval between $\underline{t^*}$ and $\overline{t^*}$ is lower than δ, the spatial dimensions usually need less refinement. In essence, bisection along the time axis is primarily driven by the need to narrow the bounds of t^*, while subdivision of the spatial domain is employed to resolve indeterminate configurations. Details and pseudo-code are given in Section 5.1.

To account for numerical errors, we employ interval arithmetic. The value of parameter δ allows for a trade-off between computation time and the accuracy of the estimation; however, regardless of the parameter choice, the algorithm always provides a conservative estimation.

Example. Consider the quadratic element in Figure 4(a). It is valid at $t = 0$, becomes invalid at some intermediate time $t^* < 0.5$, and then returns to a valid state at a later time, remaining valid until $t = 1$.
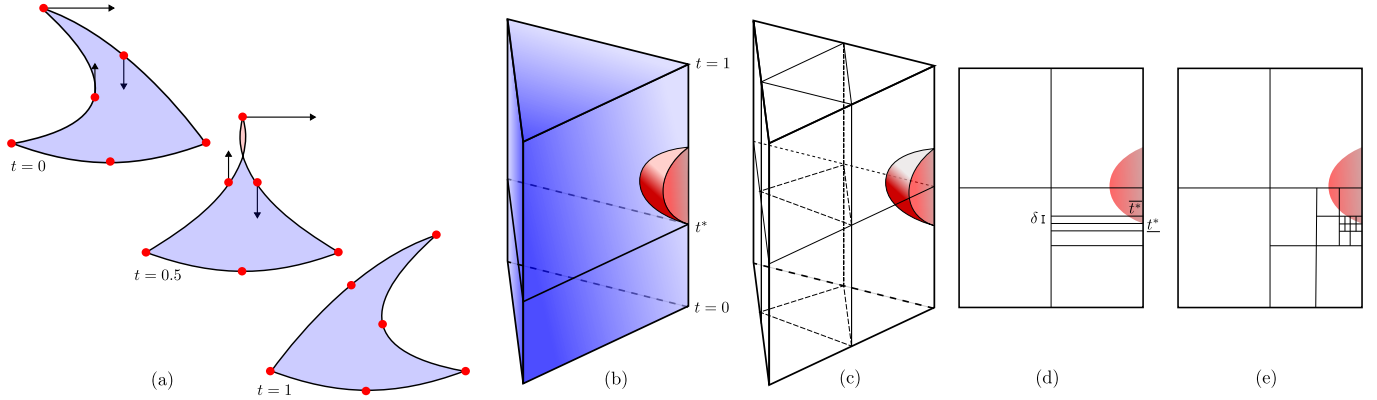
Fig. 4. (a): A 2D quadratic element with linear trajectories is valid at the start and end positions but locally inverted during the transition. (b): Its parametric domain is an upright triangular prism, where the vertical axis represents time and each horizontal slice represents the space domain at a given time. The red region denotes the portion of the domain in which the determinant $|J_x|$ is not positive. The dynamic element's Jacobian determinant is a trivariate polynomial in ξ_1, ξ_2, t of order 2 in its space variables and order 2 in the time variable. (c): The root finding algorithm bisects the time domain and quadrisects the spatial domain to isolate a thin slice $[\underline{t^*}, \underline{t^*} + \delta]$ containing the critical time of inversion t^*. In the first step, both space and time dimensions are refined, generating eight sub-domains, which are pushed into a priority queue. (d): For clarity we show a side view for the following steps. Only the time dimension is bisected: the analysis of sub-domains that do not intersect the invalid region increases the value of $\underline{t^*}$ while the analysis of sub-domains that do intersect the invalid region decreases the value of $\overline{t^*}$ until convergence. (e): If all dimensions were refined at all steps, many more sub-domains would be generated, thus killing performance.

The parameter domain for this dynamic element can be visualized as in Figure 4(b) with an upright triangular prism, where each horizontal slice of the prism represents the spatial domain at a specific time, and time progresses along the vertical axis from 0 to 1. Notably, the element inversion occurs within a localized wedge (red volume in the figure) during the times when the element is invalid. In general, inversions can occur anywhere within the domain, including regions away from vertices.

The minimum inclusion function is evaluated first for the whole prism. The result is an interval that contains zero, thus the domain is subdivided as in Figure 4(c) along both the time and the space dimensions, and eight sub-prisms are pushed to the priority queue. While processing the four prisms corresponding to the time interval $[0, 0.5]$, the minimum inclusion function returns a strictly positive interval for three of them, which are discarded from the queue; and it returns a strictly negative interval for the fourth one, which intersects the red wedge: this domain is bisected only in the time dimension and its two children are pushed onto the queue. Sub-domains spanning earlier times are processed first.

In the subsequent refinements (Figure 4(d), side view), the minimum inclusion function will always be strictly negative, hence only bisection in the time dimension will occur: the analysis of valid intervals that do not intersect the red wedge contributes to increasing the value of $\underline{t^*}$ while the analysis of intervals that do intersect the red wedge contributes to decreasing the value of $\overline{t^*}$, until convergence. Note that, if the domain were always subdivided along space and time, as in Figure 4(e), many more subdomains would be generated and the algorithm could become much slower. For the 3D order 3 armadillo dataset (Table 2, Figure 9), with 10% target error, the naive approach has an average processing time of $\sim 7ms$ per element, whereas our approach takes $\sim 230\mu s$ per element on average. For higher element orders, or queries with higher precision, the maximum number of subdivisions used in the naive approach must be limited (otherwise the check will consume unreasonable amounts of memory), and the test can fail to produce an estimate within precision on some elements.

3 RELATED WORK

We briefly describe polynomial bases and their use in graphics, methods for checking static and continuous validity of elements, and discuss their application in FEA and meshing. We conclude with an overview of robust predicate evaluation techniques, which we use in our algorithm.

High-Order Bases and Geometry for Finite Element Analysis. Linear basis functions are often a sub-optimal choice in many contexts. For example, Schneider et al. [2022] advocate for the use of high-order bases for elliptic PDEs; Bargteil and Cohen [2014], Mezger et al. [2009], Suwelack et al. [2013], Ushakova [2011] use high-order elements for animation; and Mandad and Campen [2020] propose to use a high-order basis for parametrization. Our algorithm provides a guaranteed-conservative check for the validity of these elements, increasing the robustness of any method using a high-order basis.

A related, but distinct, concept is the use of high-order geometry, where the geometry of an element is represented using a high-order polynomial. More commonly, C^0 [Jiang et al. 2021a] or C^k geometric maps (IGA) [Cottrell et al. 2009] are used. The latter option is popular in mechanical engineering, where IGA envisions the use of the same representation for both interpolating the physical quantities and for representing the geometry.

Despite the different uses, our algorithm applies to generic polynomials, hence it can be used as is to ensure the validity of the geometric map.

Static Element Inversion Check. The special case of checking the geometric validity of a linear triangle/tetrahedron in a static setting has been solved in a seminal paper by Shewchuk [1997], where a robust predicate called ORIENT2D/ORIENT3D is introduced. This paper revolutionized mesh generation and simulation, providing a reliable, yet efficient, solution to one of the basic primitives used by meshing and simulation algorithms. To the best of our knowledge, this approach has not been extended to elements with higher order; such extension is challenging because an element may flip at certain points while remaining valid at others, as in the example shown on the right side of Figure 2.

For high-order simplicial elements, a common approach to test for element inversion consists in testing only their quadrature points [Dey et al. 2001; Gargallo-Peiró et al. 2015; Maas et al. 2012; Schneider et al. 2019]: while effective at avoiding NaNs in the integration of certain diverging elastic potentials, this approach is not conservative, leading to incorrect stresses (Figure 13). An efficient method has been introduced by Johnen et al. [2018, 2013, 2014], where the Jacobian determinant of the element is represented in Bézier form: the inversion check then reduces to testing the positivity of the Bézier coefficients as they undergo adaptive refinement. Their implementation relies on floating point arithmetic, and therefore it is not conservative and can miss inversions (we provide a numerical example in Appendix E). We take inspiration from this check and similarly use adaptive Bézier subdivision to derive a continuous test: our algorithm solves a different problem by adding the time dimension and is designed to be conservative due to its judicious use of rational and interval arithmetic.

For the special case of hexahedral elements, which are commonly used in commercial finite element analysis software, this problem has been extensively studied by Ushakova [2011]. Unfortunately, these tests are insufficient to guarantee validity but are used nonetheless due to their efficiency. Vavasis [2003] proposes a sufficient condition but does not provide a conservative algorithm that takes advantage of it. Johnen et al. [2017] propose an optimized version of Johnen et al. [2014] specifically for linear hexahedral elements, however, the approach still suffers from the same floating point issues as the original test. George and Borouchaki [2014] similarly propose a recursive subdivision-based method for static tensor product elements; we are not aware of a publicly available implementation of this method.

A radically different approach is taken by Marschner et al. [2020], where a Sum-of-Squares (SOS) relaxation is used to compute the minimum Jacobian determinant, reducing the problem to solving a sequence of small semidefinite programming problems of increasing complexity. The method offers generality and an elegant formulation, but is relatively expensive computationally (multiple SDP solves per element), and only guarantees injectivity up to numerical precision of an iterative convex solver, which might result in invalid elements.

All the methods described above are designed to handle static checks exclusively, which is insufficient for validating deforming elements over time.

Continuous Element Inversion Check. To the best of our knowledge, the only paper explicitly addressing the problem of checking the validity of an element that deforms over time is by Smith and Schaefer [2015]. They propose an algorithm to estimate the safest step before an inversion for 2D linear elements by using the closed form of the roots of a degree 2 polynomial. Their approach is however not conservative due to floating point rounding, as we show in Section 8. Furthermore, the method does not scale to linear 3D elements – where the polynomial is of order 3 and robust root finding is not trivial – nor higher order elements – where the polynomial is multivariate and closed forms for the roots may not even be available.

The problem is also discussed by Anderson et al. [2014] for high-order element remapping and by Dobrev et al. [2019] for high-order meshing, however no algorithm for the validity check is proposed. Our algorithm could be used in their setting to provide a conservative validation of their resulting elements.

High-Order Meshing. High-order meshing requires a high-order validity check to ensure that elements are valid after curving. We refer to Geuzaine et al. [2015] and Jiang et al. [2021b] for an overview of the state of the art of high-order meshing. Our contribution can be used, in its reduced form for static validity, as a provably conservative check in any of these meshing algorithms to increase their robustness.

Interval Arithmetic. According to the IEEE 754 standard, the result of a floating point operation is a rounded value of the exact result. Instead of computing a single rounded value, one can compute an interval that *contains* the exact result by simply rounding in both directions. Replacing FP numbers with intervals slows the calculations by an order of magnitude on average but, on the other hand, enables a provably correct implementation of geometric algorithms [Snyder 1992]. Intervals are provided by many existing libraries such as CGAL [Brönnimann et al. 1998], Boost [Schling 2011], Filib and Filib++ [Lerch et al. 2006], BIAS [Knuppel 1994] and GAOL [Goualard 2005]. We point the reader to Tang et al. [2023] for a detailed comparison of these tools. When full portability is not required, modern SIMD architectures can be exploited to speed up interval arithmetic significantly. The basic idea is to store both the bounds in a single register that can host 128 bits, which is the space required by two double precision FP numbers, and then perform each operation on the entire register simultaneously. This approach, first introduced by Lambov [2008], is employed in the numeric kernel of the indirect predicates library [Attene 2019], which we use in our implementation.

We use intervals to represent inclusion functions and to guarantee that any possible rounding error is always tracked so as to provide a correct polynomial evaluation.

4 PRELIMINARIES AND NOTATIONS

We address n-dimensional ($n = 2, 3$) high-order meshes consisting of elements of various types. The geometry of every element is defined by a polynomial map. We will refer to the *order p* of a polynomial as the maximum exponent of a single variable (as opposed to the usual notion of degree). Table 1 gives a summary of symbols defined in the following and used throughout.

Table 1. Symbols used in the text and where they are defined.

Symbol	Meaning	Def.
n	dimension of element and embedding space	4
s	dimension of simplicial part of element	4.1
σ, σ_s^n	static reference element (with dimensions)	4.1
$\xi_1, \dots \xi_n$	spatial coordinates	4.1
p	order of an element / polynomial	4.2
x	geometric map of an element	4.2
$|J_x|$	Jacobian determinant of x	4.2
$\mathbb{I}$	intervals on the real line	4.3
$\Box f$	inclusion function for f	4.3
$\Box_{\min} f$	minimum inclusion function for f	4.3
t	time coordinate	5
$\bar{\sigma}, \bar{\sigma}_s^n$	dynamic reference element (with dim.)	5
$\bar{x}$	dynamic geometric map of an element	5
t^*	minimum time at which $|J_{\bar{x}}|$ vanishes	5
$\underline{t^*}, \overline{t^*}$	lower and upper bounds to t^*	5
δ	user-specified accuracy	5
$l_{\max}$	maximum level of recursion	5
ψ^-, ψ^+	time-only subdivision maps	5
ψ^q	q-th subdivision map	5
C_σ^p	subset of indices of corners of element σ	6
$\mathbf{T}_{\mathcal{B}\to\mathcal{B}}^\pm$	time-only subdivision matrix	6
$\mathbf{T}_{\mathcal{B}\to\mathcal{B}}^q$	q-th subdivision matrix	6
Γ_σ^p	set of domain points of order p on σ	A
γ_i	domain point	A
$\mathcal{I}_\sigma^p$	set of indices of points of Γ_σ^p	A
$\mathcal{L}_i^m$	i-th Lagrange polynomial of order m	A
$\mathcal{B}_i^m$	i-th Bernstein polynomial of order m	A
$f^{\mathcal{L}}$	vector of coefficients in Lagrange form	A
$f^{\mathcal{B}}$	vector of coefficients in Bézier form	A
$\mathbf{T}_{\mathcal{L}\to\mathcal{B}}$	transition matrix from Lagrange to Bézier	A

4.1 Reference Element Domains

For each type of element, we define a common *reference domain* (or *reference element*) $\sigma_s^n \subset [0,1]^n$ to use as the coordinate domain.

Definition 1 (Reference Domain). *Let $n, s \in \mathbb{N}$, $1 \le s \le n$. The n-dimensional reference domain $\sigma_s^n \subset [0,1]^n$ is the locus of points with coordinates $(\xi_1, \dots, \xi_n)$ that satisfy the system of inequalities:*

$$\xi_i \ge 0 \quad \forall i \in \{1, \dots, n\}$$

$$1 - \sum_{i=1}^{s} \xi_i \ge 0 \tag{1}$$

$$\xi_i \le 1 \quad \forall i \in \{n-s+1, \dots, n\}$$

With this notation, we have a general parameter space that works for all the most commonly used FEM elements. The element σ_1^1 is the unit segment; σ_1^2 is the unit square; σ_2^2 is the standard triangle; σ_1^3 is the unit cube; σ_2^3 is the unit triangular prism; σ_3^3 is the standard tetrahedron. More generally, σ_s^n is the tensor product of a standard s-simplex with a standard $(n-s)$-hypercube[1].

---

[1]Note that, the unit segment can be seen both as a 1-simplex and as a 1-hypercube; to avoid any ambiguity, we always treat it as a simplex, so that, e.g., the unit square
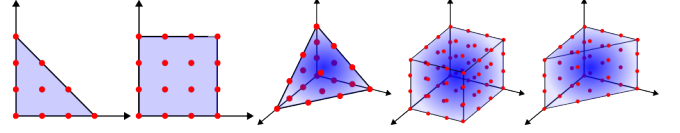
Fig. 5. Reference elements with domain points of order 3: triangle, square, tetrahedron, hexahedron, prism.
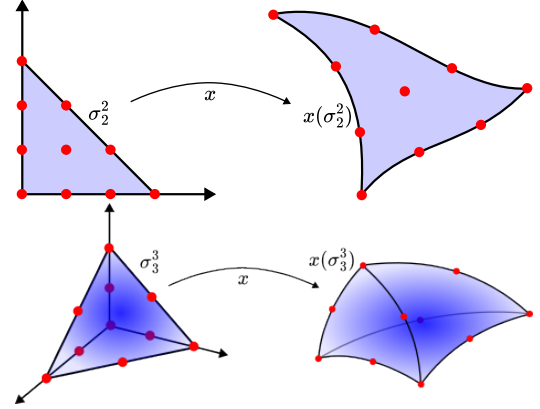

Fig. 6. Above: A domain σ_2^2 with its domain points of order 3 and the geometric map x to the physical element $x(\sigma_2^2)$ with Lagrange control points. Below: likewise for an element σ_3^3 of order 2.

4.2 High-Order Elements

A generic polynomial $f : \sigma \longrightarrow \mathbb{R}$ can be defined with a basis of polynomials: we consider here the Lagrange basis – which is most common in FEM – and the Bernstein basis that gives the Bézier form. Both representations define f as a linear combination of the basis functions with *control coefficients* associated with *domain points* that form a regular grid over the reference element; the number of domain points sets the *order p* of the polynomial (Figure 5).

For a given order and dimension, pre-computed conversion matrices allow us to convert between these two representations in both directions. The Lagrange and Bézier forms and the conversion matrices are detailed in Appendix A.

Geometric Map. The geometric map $x : \sigma_s^n \longrightarrow \mathbb{R}^n$ that maps a reference element σ_s^n into the physical element $x(\sigma_s^n)$ is represented by specifying its set of control points, where each control point is a n-dimensional point (Figure 6). Each coordinate of x is expressed with a multivariate polynomial, which is the tensor product of a s-variate polynomial of degree p with $(n-s)$ univariate polynomials of degree p, each in a different variable. It follows that the map x has order p in all its variables (even though its total degree may be higher).

In the following, we study the determinant of the Jacobian J_x of the geometric map x, denoted $|J_x|$, which is also a multivariate polynomial in the same variables as x, but of a different order. In

---

is regarded as the tensor product of a 1-simplex with a 1-hypercube, rather than as a 2-hypercube.

particular, the number of terms of $|J_x|$ rapidly increases with the dimension and order of the element. See Appendix A for details.

4.3 Minimum Inclusion Function

Informally speaking, given a real function f and a domain D, an *inclusion function* for f over D returns an interval that bounds the range of values of f in D. Inclusion functions are widely used in root finding and in the evaluation of robust predicates [Snyder 1992]. In our case, we are rather interested in an interval that just contains the *minimum* value of f.

Let $\mathbb{I}$ be the space of intervals on the real line. For $a = [\underline{a}, \overline{a}] \in \mathbb{I}$, let us define $w(a) = \overline{a} - \underline{a}$ the *width* of interval a. Let $A = a_1 \times \cdots \times a_n \in \mathbb{I}^n$ be a n-dimensional interval; we extend the definition of width as $w(A) = \max_{j=1}^n w(a_j)$. Given $D \subseteq \mathbb{R}^n$ compact, we further extend the definition of width as $w(D) = \min_{A \supseteq D} w(A)$.

Let $\Omega \subseteq \mathbb{R}^n$ be a compact domain, let f be a real function defined on Ω, and let us denote $\mathcal{P}(\Omega)$ the subsets of Ω. Given a function $f : \Omega \longrightarrow \mathbb{R}$, an *inclusion function* for f is a function $\square f : \mathcal{P}\Omega \longrightarrow \mathbb{I}$ such that, for any $D \subseteq \Omega$ we have

$$\forall \xi \in D \quad f(\xi) \in \square f(D).$$

We say $\square f$ to be *convergent* if for any $D \subseteq \Omega$

$$w(D) \to 0 \Rightarrow w(\square f(D)) \to 0.$$

In particular, if D shrinks about ξ, then $\square f(D)$ shrinks about $f(\xi)$.

DEFINITION 2 (MINIMUM INCLUSION FUNCTION). *A minimum inclusion function for the function f is a function $\square_{\min} f : \mathcal{P}(\Omega) \longrightarrow \mathbb{I}$ such that, for any $D \subseteq \Omega$ we have*

$$\min_{\xi \in D} f(\xi) \in \square_{\min} f(D).$$

If the lower end of $\square_{\min} f(D)$ is positive, we know that f is always positive in D; if the upper end is negative, we know that f has negative values, but is not necessarily negative everywhere, in D; otherwise, nothing can be said about the sign of f in D.

A convergent inclusion function can be used to find a root of a function f by subdividing the initial domain Ω until it becomes sufficiently small. Likewise, one can use a convergent minimum inclusion function to find the portions of Ω where f is positive, by recursively subdividing the domain. The type of subdivision used to perform refinement depends on the shape of Ω. For instance, while bisection can be used for a multi-interval domain, simplicial domains may require less trivial subdivision rules (Appendix B).

4.4 Interval and Rational Arithmetic

Interval arithmetic consists of a set of operations defined on the set of real intervals $\mathbb{I}$ such that if $\xi \in A_\xi \in \mathbb{I}$ and $\zeta \in A_\zeta \in \mathbb{I}$, then $(\xi * \zeta) \in A_\xi * A_\zeta$, where $*$ in the right-hand side is the interval version of the operation. If the exact result of an operation on floating point numbers falls between two representable values, rounding is required. We make our computations conservative by replacing floating point numbers with singleton intervals (i.e., intervals with matching endpoints), and rounding the left end of the result down and the right end up for every subsequent interval operation. In our code, we use the implementation of Attene [2020]. More details are provided in Appendix C.

Interval arithmetic does not prevent the propagation of error; it merely keeps track of it. One way to implement exact arithmetic is via rational numbers: as long as an algorithm only involves rational operations, we can represent numbers exactly as fractions of integer values since every floating point number is also rational. The main drawback of rational arithmetic is that it can be orders of magnitude slower than interval arithmetic because the bits needed to encode each fraction increase with computations. For this reason, our use of rational arithmetic is limited to off-line computations.

5 CONTINUOUS GEOMETRICAL VALIDITY

In a dynamic simulation, the elements of the mesh move and deform over time. Like space, time is discretized into time steps, which are typically regular. We assume that the control points move along straight-line trajectories at each time step, and, without loss of generality, we can assume each transition occurs between time $t = 0$ and time $t = 1$. Following Definition 1 we have:

DEFINITION 3 (DYNAMIC REFERENCE ELEMENT). *Let σ_s^n be a reference element. The dynamic element $\bar{\sigma}_s^n$ of σ_s^n is the $(n+1)$-dimensional reference element $\sigma_s^{n+1} = \sigma_s^n \times [0, 1]$.*

Assuming linear trajectories, the *dynamic geometric map* $\bar{x} : \mathbb{R}^{n+1} \to \mathbb{R}^n$ of order p for $\bar{\sigma}_s^n$ is expressed by linear interpolation of the n-dimensional geometric maps $x^0(\xi)$ and $x^1(\xi)$ of the static element at the two consecutive time steps:

$$\bar{x}(\xi, t) = x^0(\xi) + t(x^1(\xi) - x^0(\xi)). \tag{2}$$

This map is of order p in the ξ variables and linear in t.

5.1 Continuous Validity Test

Assuming that the input element is valid at time $t = 0$, the *continuous validity test* consists of determining whether or not the Jacobian determinant $|J_{\bar{x}}(\xi)|$ is everywhere greater than 0 on $\bar{\sigma}$ at all times in $[0, 1]$. If this is not the case, the algorithm should find the earliest time $t^* \in (0, 1]$ in which the element becomes invalid – hence, it is valid within $[0, t^*)$.

Target accuracy. Since finding an exact solution is not necessary and very expensive, we instead settle for a conservative estimate $\underline{t}^*$ that is close to t^* up to a given user-provided threshold δ, i.e. $t^* \in [\underline{t}^*, \underline{t}^* + \delta]$. Moreover, since we assumed that $t^* > 0$, we require that $\underline{t}^*$ be strictly positive as well, regardless of the value of δ. This threshold is used to trade off accuracy and time performance.

Early termination. An intrinsic challenge of both the static and dynamic problems is that certifying the positivity of a polynomial can be arbitrarily hard. Therefore, there is no upper bound on the number of subdivisions required to assess the validity of an element.

To prevent the algorithm from taking unreasonable time, we employ a termination criterion that triggers when refinement becomes excessive. In these extreme cases, we halt and provide an estimate $\underline{t}^*$ that may not be within δ of the true value t^*, but is nevertheless conservative.

Let the *depth* of a subdomain S be the number of subdivisions required to obtain S from the initial domain. In our implementation we stop when the depth of a subdomain S exceeds a threshold $l_{\max}$. How often this condition is triggered in practice depends on the

dataset and element type. In our benchmark, choosing a value of $l_{\max} = 7$ was enough to reach the desired precision of 10^{-2} on all but very few 3D tensor product elements. We refer to Table 2 and Section 9 for details.

Minimum inclusion function. We rely on a convergent inclusion function, which comes together with a procedure to decompose an element into sub-elements. For the sake of clarity and generality, we first describe our algorithm avoiding the details on how we define our inclusion function and domain subdivision strategy, which we detail in Section 6. We also use $J = |J_{\bar{x}}|$ as a short-hand notation for the determinant of the Jacobian of the dynamic element at hand.

Given a generic inclusion function $\Box J$ on the domain $\bar{\sigma}_s^n$, we define a minimum inclusion function as follows: let $\Box J(D) = [\underline{J_D}, \overline{J_D}]$ then

$$\Box_{\min} J(D) = [\underline{J_D}, \min_{\bar{\xi}_i \in \mathcal{D}_D} J(\bar{\xi}_i)],$$

where $\mathcal{D}_D$ is a small set of samples in D. In practice, we sample J at these points to bound the minimum of J from above. Note that, if J is negative at any of those samples, we know that the element becomes invalid in D.

Subdivision Maps. Given a reference domain $\sigma = \sigma_s^n$, we define a set of Q linear maps $\{\psi^q : \sigma \longrightarrow \sigma\}_q$ called the *subdivision maps* of σ, and we call $\psi^q(\sigma)$ a *subdomain* of σ; we require that the union of all subdomains is σ, and the intersection of any two subdomains is either empty or has dimension less than n. We define the standard subdivision maps for $n \in \{2, 3\}$ that we use in our implementation in Appendix B. In the following, we always assume that $Q = 2^n$.

For a time dependent reference domain $\bar{\sigma} = \bar{\sigma}_s^n$, its subdivision maps are the same as σ_s^{n+1}, but we also define two additional *time subdivision maps*, denoted ψ^- and ψ^+, such that $\psi^-(\bar{\sigma})$ and $\psi^+(\bar{\sigma})$ are respectively the lower and upper half of $\bar{\sigma}$ when bisected in the time dimension only.

Pseudo-Code. The pseudo-code of the algorithm is given in Algorithm 1. The algorithm takes in input a polynomial $J(\xi, t)$ defined on domain $\bar{\sigma}$ and the thresholds δ and l_{max} and returns a time $\underline{t^*}$ without invalid configurations, and within a time δ of an invalid configuration. The algorithm keeps internal current lower and upper bounds for t^*, initializing them to 0 and 1, respectively. Given a subdomain $S \subset \bar{\sigma}$, pseudocode functions $\text{STARTTIME}(S)$ and $\text{ENDTIME}(S)$ return respectively the minimum and maximum values of the time coordinate for points in S.

The algorithm uses a priority queue P (line 2) of subdomains of σ, with a related priority function $\prec$ (line 28) giving higher priority to sub-domains that span intervals of time with an earlier start point (that is, a lower minimum t). The initial domain σ is pushed into the priority queue P (line 5); then elements are popped from the P one by one (line 13), and if their minimum inclusion function does not guarantee their validity, they are subdivided and their subdomains are pushed to P. See the next paragraph for the subdivision strategy. This continues until the queue becomes empty or an early exit condition is met. By construction, the priority function $\prec$ guarantees that when we pop an element S from the queue, then J is positive at all times before $\text{STARTTIME}(S)$, and $\underline{t^*}$ can then be updated accordingly.

Algorithm 1 Maximum valid time step with inclusion functions

```
 1: function MAXVALIDSTEP(J, δ, lmax)
 2:     P ← PRIORITYQUEUE(≺)          ▷ priority queue for subdomains
 3:     t̄* ← 1                        ▷ initialize upper bound of t*
 4:     t* ← 0                        ▷ initialize lower bound of t*
 5:     PUSH(P, σ)
 6:     F ← FALSE      ▷ flag of whether an invalidity has been found
 7:     l ← 0              ▷ maximum subdivision depth reached so far
 8:     while TRUE do
 9:         if F ∧ (t̄* − t* ≤ δ) ∧ (t* > 0) then ▷ reached accuracy
10:             return t*               ▷ conservative estimate of t*
11:         if ISEMPTY(P) then
12:             return 1
13:         S ← POP(P)             ▷ get the next subdomain from P
14:         l ← max{l, DEPTH(S)}         ▷ update maximum depth
15:         if l > lmax then     ▷ maximum level reached: give up
16:             return t*            ▷ conservative estimate of t*
17:         t* ← STARTTIME(S) ▷ everything before this time is valid
18:         I ← □min J(S)             ▷ check minimum inclusion
19:         if HIGH(I) ≤ 0 then       ▷ there is an invalidity in S
20:             if ENDTIME(S) < t̄* then
21:                 F ← TRUE
22:                 t̄* ← ENDTIME(S)
23:             PUSH(P, ψ⁻(S))          ▷ bisect on the t axis only
24:             PUSH(P, ψ⁺(S))          ▷ bisect on the t axis only
25:         else if ¬(LOW(I) > 0) then
26:             for q ∈ {1, ..., Q} do
27:                 PUSH(P, ψq(S))  ▷ subdivide on ξ and bisect on t
28: function ≺(S0, S1)                       ▷ priority function
29:     if STARTTIME(S0) ≠ STARTTIME(S1) then ▷ lower time first
30:         return STARTTIME(S0) < STARTTIME(S1)
31:     else    ▷ for ties, prioritize boxes most likely to be invalid
32:         return HIGH(□min J(S0)) < HIGH(□min J(S1))
```

Early exits occur if the required accuracy δ is achieved (line 9), meaning that the difference between $\overline{t^*}$ and $\underline{t^*}$ is less than δ, or the maximum depth $l_{\max}$ has been reached (line 15), meaning that we pop from the queue an interval that comes from a sequence of $l_{\max}$ subdivisions.

Subdivision Strategy. If the interval I returned by $\Box_{\min} J(S)$ is completely negative – meaning that S contains negative values of J (line 19) – then the upper bound $\overline{t^*}$ is updated, and element S is bisected along the time dimension only, with the two resulting subdomains being pushed into P. Conversely, if interval I contains zero, the element S is split along all its dimensions (line 25), including time, according to the subdivision scheme of reference element $\bar{\sigma}$; again, the resulting elements are pushed into P. Finally, no subdivision is necessary if I only contains positive values, and the space-time region occupied by S will not be considered again for the remainder of the algorithm.

Note that, by bisecting only the time dimension (line 19), we postpone any refinement of the spatial dimensions until we find a time interval in which J may potentially be positive everywhere. This strategy allows us to avoid many unnecessary refinements

in the space dimensions, and to decouple the subdivision on time (controlled by accuracy δ) from the subdivision in space (which does not have an accuracy requirement).

6 IMPLEMENTATION

The implementation of our method requires designing a minimum inclusion function $\square_{\min} f$ and a corresponding subdivision strategy that uses robust computations while keeping the runtime sufficiently low to enable its use within a simulation loop.

6.1 Inclusion Functions for Space and Time

As observed by Snyder [1992], interval arithmetic provides a universal way to design inclusion functions. Given a polynomial $f : \Omega \longrightarrow \mathbb{R}$ and $D \subseteq \Omega$, let A_D be the smallest multi-interval containing D. We could define

$$\square f(D) = f(A_D),$$

where the evaluation of f on the right side is intended with interval arithmetic, and thus returns an interval. Any strategy subdividing D and reducing A_D (e.g., bisection along all coordinates) provides a convergent inclusion function.

We tried this approach, but the inclusion functions may be very loose about f and require many refinement steps to converge, or even get stuck on nearly invalid elements due to the numerical error accumulating too fast for the inclusion function to keep up with. We compare the time performance for the static case only in Table 3. We instead follow the approach proposed by Johnen et al. [2014] for the static validity test and extend it to our continuous setting.

Overview of Bézier Refinement. Let f be the order m polynomial of which we want to find the minimum on $\bar{\sigma}$ (in our case, $f = |J_{\bar{x}}|$).

Our inclusion function is based on the Bézier representation of f and a recursive decomposition of $\bar{\sigma}$. The reason why we want to represent our polynomial in the Bézier basis is the convex hull property [Farin 2001], by which the values of f on $\bar{\sigma}$ are bounded by the minimum and maximum coefficients of f when expressed in the Bézier basis.

To obtain the vector of Bézier coefficients $f^{\mathcal{B}}$ of f, we first compute its vector of Lagrange coefficients $f^{\mathcal{L}}$, which can be obtained by simply evaluating f at the domain points; then we premultiply $f^{\mathcal{L}}$ with a change of basis matrix $\mathbf{T}_{\mathcal{L} \to \mathcal{B}}$ that we shall call *transformation matrix*, which is described in detail in Appendix A.

Let $\bar{I}_{\sigma}^m$ be the set of indices of the control points of f and $\bar{C}_{\sigma}^m \subset \bar{I}_{\sigma}^m$ be the set of indices corresponding to the corners of $\bar{\sigma}$ at time 1. Since the Bézier basis is interpolating at the corners of the domain (i.e. $\beta_j = f(\gamma_j)$ for all $j \in \bar{C}_{\sigma}^m$), we define the minimum inclusion function as

$$\square_{\min} f(\bar{\sigma}) = [\min_{i \in \bar{I}_{\sigma}^m} \beta_i, \min_{j \in \bar{C}_{\sigma}^m} \beta_j]. \tag{3}$$

Therefore, if all entries of $f^{\mathcal{B}}$ are positive we know the element is valid everywhere, and if any of the corner entries is non-positive we know that the element is invalid at the end time. Otherwise, the interval returned by the inclusion function contains zero, and we need to refine the search by subdividing $\bar{\sigma}$.

The subdivision of $\bar{\sigma}$ is performed via another set of change of basis matrices, dubbed *subdivision matrices*, $\mathbf{T}_{\mathcal{B} \to \mathcal{B}}^q$, for $q = 1, \ldots, Q$.

Premultiplication of $f^{\mathcal{B}}$ by these matrices gives a Bézier representation of f on a smaller portion of the domain, which can be used to compute tighter bounds local to each subdomain. These matrices are defined in Appendix A.

Time-only refinement. Bisection in the time dimension only is performed analogously by multiplication of $f^{\mathcal{B}}$ with two *time subdivision matrices* $\mathbf{T}_{\mathcal{B} \to \mathcal{B}}^-$ and $\mathbf{T}_{\mathcal{B} \to \mathcal{B}}^+$, also described in Appendix A.

6.2 Robust Computation

Rational Precomputation of matrices. All the transformation and subdivision matrices $\mathbf{T}_{\mathcal{L} \to \mathcal{B}}$, $\mathbf{T}_{\mathcal{B} \to \mathcal{B}}^q$, and $\mathbf{T}_{\mathcal{B} \to \mathcal{B}}^{\pm}$ are only dependent on element type (tetrahedron, hexahedron, etc.) and order, and as such can be precomputed offline. Since we want to minimize the accumulation of error in our computations, and all entries of these matrices are rational, we construct these matrices using exact rational arithmetic for each element type and order. The outcome is a rational matrix, which we convert to intervals by rounding the two endpoints outward if the exact value cannot be represented as a floating point number. The resulting interval is guaranteed to contain the exact value of the fraction while being as tight as possible.

Intervals. The input to our check is the set of Lagrange control points for the elements of a mesh, represented in floating point. Each floating point coordinate is converted to a singleton interval (i.e. an interval with zero width) and all subsequent operations are performed in interval arithmetic with conservative rounding.

Summary. The combination of rational precomputation and interval arithmetic ensures that our algorithm is conservative while maintaining a low computational cost (Section 8): the rational precomputation is performed only once offline and does not affect runtime, while the use of interval arithmetic adds a minor ($\sim 2\times$) overhead over a direct floating point implementation.

6.3 Acceleration

Global Queries. In practical applications, one is often interested in the maximum time for which *all* elements are valid. We refer to this as a *global* dynamic query and give a strategy to accelerate it.

After an invalid element has been found (with estimated valid time step $\underline{t}^*$), it becomes unnecessary to validate the other elements at later times: $\underline{t}^*$ will anyhow be the maximal allowed step. It might be tempting to terminate early as soon as an inverted element has been found. However, this is not conservative as some other elements may still have a lower $\underline{t}^*$. We thus keep track of the smallest value of $\underline{t}^*$ found in previous checks and leverage the fact that $\underline{t}^*$ can never decrease in Algorithm 1 to stop computation on an element as soon as its estimate for t^* exceeds the running minimum.

The order in which elements are processed matters for global queries: it is beneficial to process elements that are most likely invalid first, as it will provide higher opportunities for this pruning strategy to be effective. For this reason, we first sort the polynomials according to their constant term, in ascending order. For the order 3 Armadillo mesh in Figure 9, this strategy improves the total running time by about 60% over running the queries individually; whereas the speedup is less relevant for the order 2 mesh (about 12%).

Parallelization. Validity checks for meshes are trivially parallelized by processing elements in batches. To avoid any synchronization between different threads, every batch of queries assigned to a thread i does an independent sorting and keeps its own running minimum $\underline{t}^*_i$ to use as an early termination condition, as explained in the previous paragraph.

Precomputation of Jacobian determinant. The input to the subdivision procedure is a Lagrange representation of the Jacobian determinant polynomial of the element. This only depends on the shape and order of the element, as well as its control points.

For each element type and order combination, we symbolically compute the expression of each Lagrange coefficient in terms of the control point coordinates, and remove common subexpressions with CSE [Muchnick 1997] (we use the implementation in SymPy [Meurer et al. 2017]). This approach increases the compilation time but provides dramatic runtime performance boosts: for the order 3 Armadillo dataset, we get a speedup of about 20×.

7 APPLICATION TO SIMULATION

Incremental potential time-stepping [Kane et al. 2000] is becoming popular in graphics [Li et al. 2020] and biomechanics [Martin et al. 2024] due to its robustness to extreme deformation and contact[Chen et al. 2022; Fang et al. 2024, 2021; Ferguson et al. 2023, 2021; Huang et al. 2024a,b; Lan et al. 2022a, 2023, 2022b, 2021; Li et al. 2021, 2023a, 2022, 2024; Shen et al. 2024]. We briefly summarize the approach here, without contact handling, as it is relevant to motivate the need for a continuous dynamic positivity check in physical simulation: as part of this overview, we will show that the check alone is insufficient, as IPC also requires a *consistent* invalidity-aware quadrature rules, which we introduce in Section 7.1.

7.1 Continuous Validity in Simulation

The updated displacement u^{t+1} of an object at the next time step is computed solving an *unconstrained* non-linear energy minimization:

$$u^{t+1} = \arg\min_u E(u, u^t, v^t), \tag{4}$$

where u^t is the displacement at the step t, v^t is velocity, and $E(u, u^t, v^t)$ is a time-stepping Incremental Potential [Kane et al. 2000]. We refer to Li et al. [2020] for more details.

For common non-linear material models, this potential is infinite when an element has a negative Jacobian, as the Jacobian determinant appears in the denominator of the expression. A physically valid trajectory cannot reach a state with infinite potential: however, this is a challenging condition to enforce in practice.

Line Search. The potential E is minimized with a descent algorithm (gradient descent or Newton), which computes a local approximation of a descent direction: this approximation might, for a finite step length, cross a region with infinite potential (Figure 7). This is a typical challenge in collision detection [Wang et al. 2021b], but is rarely considered for the elastic potential – the only work we know that considers this problem, in 2D only, is Smith and Schaefer [2015]. This challenge can be solved using a continuous inversion check within the line search, which is the focus of our work. To the best of our knowledge, state-of-the-art IPC solvers Li et al. [2020]
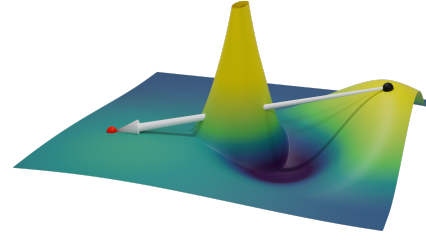

Fig. 7. Potential landscape and one descent step from the black point. The descent step to the red point is invalid since it crosses an invalid region with infinite potential.
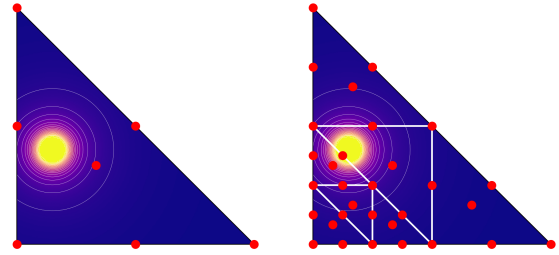

Fig. 8. The function is infinite in the bright yellow region, and its integral over the whole triangle is also infinite. Numerical integration using a fixed quadrature rule (left) erroneously produces a finite value. Our adaptive quadrature technique (right) puts quadrature points in the infinite-valued region and correctly captures the behavior of the function.

and Schneider et al. [2019] use a static check instead of a continuous one, which cannot guarantee trajectory validity.

Quadrature. Non-linear elastic potentials cannot be integrated exactly with numerical quadrature (as they are not polynomials), leading to unbounded errors for diverging potentials. We show in Figure 8 an example of an element with an infinite potential integrated with both a standard fixed quadrature rule and the adaptive quadrature derived by our algorithm: only in the second case does the numerical integration correctly diverge. The use of a fixed quadrature leads to solver failures as the direction computed using quadrature is not a descent direction and might thus block the progress of the solver.

7.2 Invalidity-Aware Quadrature Rules

Our algorithm can output additional information to generate adaptive quadrature schemes: The goal is to produce a set of quadrature points for the static element σ such that at least one of the points would end up in an invalid region at time t^*, correctly making the integral computed using quadrature diverge at that time.

Tracking Subdivisions. In Algorithm 1, each subdomain keeps track of the sequence of subdivisions of $\bar{\sigma}$ that were taken to produce it. However, since we only require a quadrature rule for σ and not $\bar{\sigma}$, it is sufficient to keep track of spatial subdivisions: this means that when an element is subdivided in time only using ψ^+ and ψ^-, the

sequence of the children will match that of the parent; and when an element is subdivided in all dimensions using the subdivision scheme ψ^q for $\bar{\sigma}$, pairs of subdomains that span the same region of space at different times will share the same sequence.

To produce the adaptive quadrature rule, our algorithm uses the sequence of spatial subdivisions of the element whose minimum inclusion function reduced $\overline{t^*}$ for the last time, or equivalently, the element that found the earliest invalidity. Unless the early exit condition for maximum depth is triggered, such invalidity is guaranteed to be in $[\underline{t^*}, \underline{t^*} + \delta]$. In the very rare cases when the algorithm fails to find an invalid point and gives up earlier, we instead return the subdivision sequence of the element with the deepest hierarchy, which is likely to be very close to an invalid region. In this case, the optimization step will still be guaranteed to be valid for its duration.

Adaptive Quadrature. This information is then used to partition the static element by recursively subdividing it using the very same sequence of subdivisions, as in Figure 8 (right). A standard quadrature rule is applied to every subelement of σ, and the integral is evaluated as the sum of integrals on all subdomains. However, in order to guarantee that the newly placed quadrature points would indeed intersect an invalid region in the full time step, it is required that the quadrature rule contain all points in the sampling set $\mathcal{D}_D$ used to compute the minimum inclusion function, projected onto σ; for example, if $\mathcal{D}_D$ is the set of corners of $\bar{\sigma}$, the selected scheme must place quadrature points at the corners of σ.

8 RESULTS

Our algorithm is implemented in C++, using PolyFEM [Schneider et al. 2019] for finite element (FE) system construction, IPC Toolkit [Ferguson et al. 2020] for evaluating IPC potentials and collision detection, Pardiso [Alappat et al. 2020; Bollhöfer et al. 2019, 2020] for the large linear systems in our global Newton solves, [Attene 2019, 2020] for interval computation, GMP [Granlund and the GMP development team 2012] for rational computation, and OpenMP [Dagum and Menon 1998] for parallelization.

The simulation experiments are run on a cluster node with an Intel Cascade Lake Platinum 8268 processor limited to 16 threads and 32Gb of memory. The benchmark experiments are run single-threaded on a laptop computer with an AMD Ryzen 7 4000 processor and 16GB of memory.

Our reference implementation, used to generate all results, and our evaluation datasets will be released as an open-source project.

8.1 Benchmark of filtered queries

Collection and Ground Truth. We collect queries of element inversion checks from elastodynamic simulation data using IPC [Li et al. 2020] and the Neo-Hookean elasticity model. We pick two 2D models and two 3D models shown in Figure 9. We bend the bar and compress the kangaroo in 2D, whereas for 3D models we twist them by $90°$ while compressing them by 20%. Note the queries included in the benchmark are a subset of the entire simulation, since the vast majority of elements are valid. For collection, we discard all queries where the initial configuration is invalid, which might happen as the positivity check in IPC [Li et al. 2020] is not conservative; in each line search in the nonlinear solves, we evaluate the Jacobian
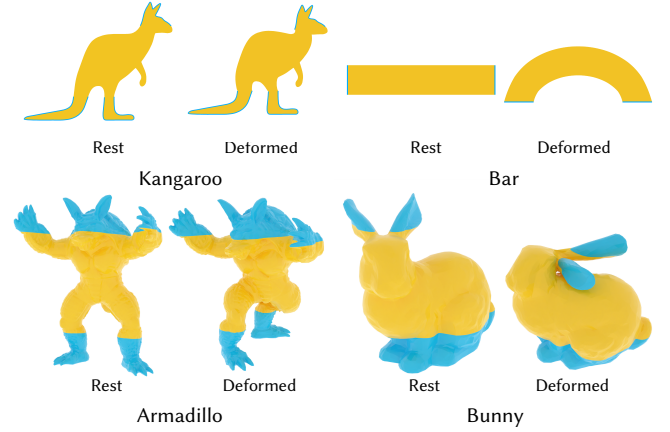

Fig. 9. Initial and final frames of simulations from which queries are exported. The parts in cyan are used as handles to twist and compress the model while the deformation occurs in the parts in yellow.

at every quadrature point J_i for $i = 1, 2, \ldots, n$ and collect elements with elements with $\min_i J_i \leq 0$ or $\min_i J_i / \max_i J_i < 0.2$.

Our benchmark contains 172000 2D queries from the Kangaroo model (orders 1 through 4), 54985 3D queries from the Armadillo model (orders 1 through 3), 19800 3D queries from the Bunny model (orders 1 through 3), 130291 queries for the Bar model of order 1, and 31879 queries for the Bar model of order 2 (Table 2).

Correctness. To validate correctness we compute the times of inversion by symbolically computing the roots of $|J_f|$ with minimum t using Mathematica [Wolfram Research 2023] (see Appendix D). We restrict the ground truth to order 1 and 2 for triangles, and order 1 for quads and tetrahedra, due to the limitations of symbolic solvers: for a higher-order basis, Mathematica could not return a result within 6 hours on some of the elements. To the best of our knowledge, ours is the first dataset containing conservative times of inversion. Our algorithm correctly detects all invalid elements and returns conservative answers for the inversion times.

Efficiency. The average per-element cost of our algorithm increases with the degree, from around $1.5\mu s$ for order 1 (both in 2D and 3D) to around $10\mu s$ for order 4 in 2D and $300\mu s$ for order 3 in 3D. Invalid elements are more expensive to process on average, as they require subdivisions until precision is reached, whereas most valid elements can be resolved in a single iteration if all their Bézier coefficients are positive. See Table 2 and Figures 10 and 11 for details.

When restricted to the static case (i.e., check the validity of an element at a given time), our algorithm becomes considerably faster than the dynamic one (by a factor of about $50\times$ on the order 3 Armadillo), and it has a running time slightly faster ($2\times$) than the non-conservative static baseline used in PolyFEM and FEBio, which consists of checking the sign of $|J_x|$ obtained with (inexact) floating point computations only at quadrature points thanks to our optimizations. See Table 3 for details.

Table 2. Results of our continuous validity test on 2D and 3D datasets element types and orders. We report the number of elements (total, valid on the whole interval, invalid at some time) and processing time in microseconds per element (average, average for valid elements, average for invalid elements, median, maximum, standard deviation). On some datasets, marked with an asterisk (*), the algorithm "gives up" on very few elements and returns a conservative answer: see details in Section 9. In all other cases, the algorithm reaches the target precision of 1% on all elements.

Dataset	Element	n	s	p	Element count			Time per element (μs)					
					tot	val	inv	avg	avg val	avg inv	med	max	std
Kangaroo	Triangles	2	2	1	172800	146248	26552	1.60	1.32	3.10	1.40	205.40	0.85
				2	172800	145602	27198	2.14	1.69	4.54	1.89	42.53	1.10
				3	172800	145623	27177	3.83	3.10	7.75	3.28	67.26	1.86
				4	172800	145613	27187	9.28	7.94	16.45	7.96	309.19	4.03
Bar 2D	Quadrangles	2	1	1	112887	14816	98071	3.93	2.95	4.07	3.84	10203.74	34.23
				2	83653	75085	8568	74.67	7.89	659.96	2.93	2046480.70	10296.53
Armadillo	Tetrahedra	3	3	1	54985	51605	3380	1.56	1.44	3.43	1.47	21.58	0.53
				2	54985	48599	6386	9.90	7.34	29.41	5.31	6726.90	40.37
				3	54985	47988	6997	362.79	130.92	1953.05	44.00	78411.99	1593.81
Bunny	Tetrahedra	3	3	1	19800	19561	239	1.47	1.45	3.51	1.40	49.38	0.63
				2	19800	19058	742	7.27	5.91	42.22	5.31	1729.90	24.56
				3	19800	18954	846	175.41	65.97	2627.34	43.02	60907.39	1294.73
Bar 3D	Hexahedra	3	1	1	56031	24918	31113*	325.33	11.10	576.99	16.55	6650064.32	39487.14
	Prisms	3	2	1	82403	56913	25490*	246.64	8.22	778.96	2.38	2407415.47	16401.57

8.2 Comparisons

We are not aware of any other algorithm that provides a continuous validity check for elements of arbitrary order, so we compare it with algorithms that solve only a subset of the problem.

Linear Continuous. For the special case of triangular elements with linear basis, Smith and Schaefer [2015] propose to use a symbolic solver to find the roots of $|J_f|$. While extremely efficient ($1\mu s$ on average), this approach can fail to produce correct results due to numerical errors: on the 26552 invalid elements in the linear Kangaroo dataset (Figure 9), their method fails to detect inversions 13324 times, producing a time step larger than the ground truth maximum. Attempting to be conservative with a "large" numerical threshold of 10^{-5} still fails in 1563 tests (~5%).

This method is limited to linear triangles, and cannot be extended to other elements or degrees due to its reliance on closed-form expressions for the roots.

Static High Order. For the special case of static validity check for elements of arbitrary type and order, Johnen et al. [2014] introduce a method based on adaptive subdivision. In the static case, our solution implements the same algorithm, but with robust arithmetic and additional pre-computations for the Jacobian and transformation matrices. When implemented with double precision floating point arithmetic, this method is more efficient than our static approach (on the 3D Armadillo model of order 3 our method takes $3\mu s$ longer per element on average, see Table 3), however it is not robust. We provide one example of an inverted element incorrectly detected as valid by the floating point implementation in Appendix E. Code is available at https://gitlab.com/fsichetti/hocgv.

Table 3. Comparison of methods for static validity checks, performed at $t = 1$ on the order 3 Armadillo dataset. We list the number of detected valid and invalid elements, the number of elements for which the test was undecided, and the average and median computation times per element. Sampling at quadrature points is fast, but incorrectly classifies several invalid elements as valid; bisection with a robust interval-based inclusion function is fast on "easy" elements, but struggles a lot with nearly-inverted elements and fails to classify several of them; our implementation of the Bézier refinement based inclusion check by Johnen et al. [2014] and our conservative method give the same results; our method is guaranteed correct at a slight performance cost. Our precomputations (last two rows) decrease the computation time by at least an order of magnitude.

Static algorithm	#val	#inv	#und	avg μs	med μs
Quadrature Points	48214	6771	-	16	16
Interval Bisection	46281	6561	2143	1400	23
FP Bézier (no optim.)	48050	6935	0	71	78
Ours (no optim.)	48050	6935	0	86	95
FP Bézier	48050	6935	0	5	5
Ours	48050	6935	0	8	8

8.3 Elastodynamic Simulation

We integrate our check within the PolyFEM software [Schneider et al. 2019] and use it to reproduce a selection of the bundled elastodynamic simulations. We report our findings as we integrated the check, since they highlight some fundamental issues with existing high-order FE solvers and the non-linear material models commonly used in graphics and engineering.

Potential Formulation. In the simulation, we minimize the Neo-Hookean energy with Newton's method. The Neo-Hookean energy density has the form of

$$w_e(F) := \frac{\mu}{2}(\text{Tr}[FF^T] - 2 - 2\log(\det F)) + \frac{\lambda}{2}\log^2(\det F), \quad (5)$$

Table 4. Simulation Statistics. Columns from left to right: simulation dimension, geometric and solution bases orders, number of cells, number of time steps, peak memory usage, the average time of the simulation per time step, the average time of our check per time step, the average time of each check over the entire mesh, and average time of the simulation per time step using the quadrature point check in PolyFEM instead of ours (baseline)

		Order				Mem	Timing (sec)			
	dim	geom	soln	# Cells	# Steps	(GB)	total / step	check / step	check / query	total / step (baseline)
Beam-twist	3	1	2	1740	400	1.4	16	5.7	0.067	7.6
Ring-twist	2	1	2	1136	100	0.7	0.72	0.16	0.011	0.15
Mat-twist	3	1	2	2166	250	1.3	10	2.9	0.069	6.9
Armadillo-rollers	3	1	2	5978	230	3.3	114	27	0.19	111
Microstructure	3	4	2	6414	25	2.9	531	472	5.6	52

where F is the deformation gradient matrix (2×2 or 3×3), λ and μ are Lamé parameters. Call J_d and J_r the Jacobians of the deformed element and rest element respectively, the deformation gradient is

$$F = J_d (J_r)^{-1}.$$

Suppose the rest element is valid, i.e., $\det J_r > 0$; then the positivity of $\det F = \det J_d / \det J_r$ depends solely on $\det J_d$, i.e. the Jacobian determinant that we are checking. As long as $\det J_d > 0$, Equation (5) is valid.

Baseline Simulator. Our baseline simulator is PolyFEM [Schneider et al. 2019], using the convergent IPC formulation [Li et al. 2023b] on tetrahedral meshes, quadratic Lagrangian bases and the Neo-Hookean material model [Ogden 2013]. PolyFEM uses only a static inversion check on the quadrature points: in Figures 1 and 12 we show that the final result contains many invalid elements leading to NaN in stress (Figure 13A). These invalid elements are not detected using the check on quadrature points alone, but are correctly identified by our conservative check (Figure 13B).

Conservative Line-Search Only. Replacing the static sampling in-validity check with our conservative and continuous check without adaptive quadrature leads to convergence issues in the solver. When the elements become close to inversion, the negative gradient direction is not a descending direction. This happens due to the error in the numerical integration of the potential, that can be solved by using our adaptive quadrature method. All the simulations in our experiments suffer from this issue: the simulation halts after several steps (twist-beam fails at time step 145, armadillo-rollers at 33, mat-twist at 141, ring-twist at 28, and the microstructure at 7).

Conservative Line-Search + Adaptive Quadrature. This solution works but dramatically increases the number of non-linear iterations needed in the Newton method. The remaining issue is that, while the Neo-Hookean energy density is infinite at the point where the Jacobian determinant is zero, it grows at a slow rate. Mathematically, when the Jacobian determinant is exactly zero at one point and positive everywhere else, the integral of the Neo-Hookean energy still may be finite: consider the integral of a 1D function $f(x) = -\log(|x|)$ on the interval $[-1, 1]$, although it has a singularity at $x = 0$, the integral is

$$\int_{-1}^{1} -\log(|x|) dx = -2 \int_{0}^{1} \log(x) dx = 2 < \infty$$

This is undesired since it leads to NaN in Equation (5) and stress evaluation, causing failure of the simulation. This problem can be mitigated by adding an additional barrier term $\frac{\mu}{\det F}$ to Equation (5). While this solution formally fixes the problem (and also practically fixes it in our experiments), it makes the potential harder to minimize while changing the material model: we are not aware of any analysis of this problem, and we believe it is an exciting avenue for future work.

Observations. With these modifications, PolyFEM produces results without invalid elements. We note that given that the material models are infinite for invalid elements, the existing non-conservative approaches often create solutions that are non-physical. A more detailed study of the effect of these errors could be an interesting venue for future work.

A second observation is that the issue with the slow growth of the Neo-Hookean potential (and others, such as Mooney-Rivlin) is likely due to the misuse of these models for deformations that are outside of the regime they are designed to handle. It would be interesting to carefully study experimentally how accurate these material models are under extreme deformations, and see if it is possible to design other material models which are numerically more suitable for interior point optimization.

High-Order IPC. We reproduce simulation examples that use high-order finite elements in Ferguson et al. [2023], including Mat-twist, Armadillo-rollers, and Microstructure (Figure 12). In Figure 1, we include two more examples: Beam-twist and Ring-twist. A video of these simulations is provided as additional material. In the Beam-twist, we apply Dirichlet boundary conditions on the two sides of the beam, rotate one side, and keep the other side fixed; in the Ring-twist, we apply Dirichlet boundary conditions to rotate the inner circle of the ring with constant speed and allow the outer circle free to move. In Figure 12, the simulation results in Ferguson et al. [2023] have flipped elements since the simulator only checks Jacobian at quadrature points in each element, while with our conservative check and adaptive quadrature, there is no flipped element. We report the statistics in Table 4. For quadratic elements, the runtime of our method is at worst comparable to the solve time and can be as fast as 23% of the solve time; for quartic geometric elements and quadratic solutions, our method is much slower than the solve time, since the solve is on quadratic elements while the Jacobian check should be performed on the quartic elements.
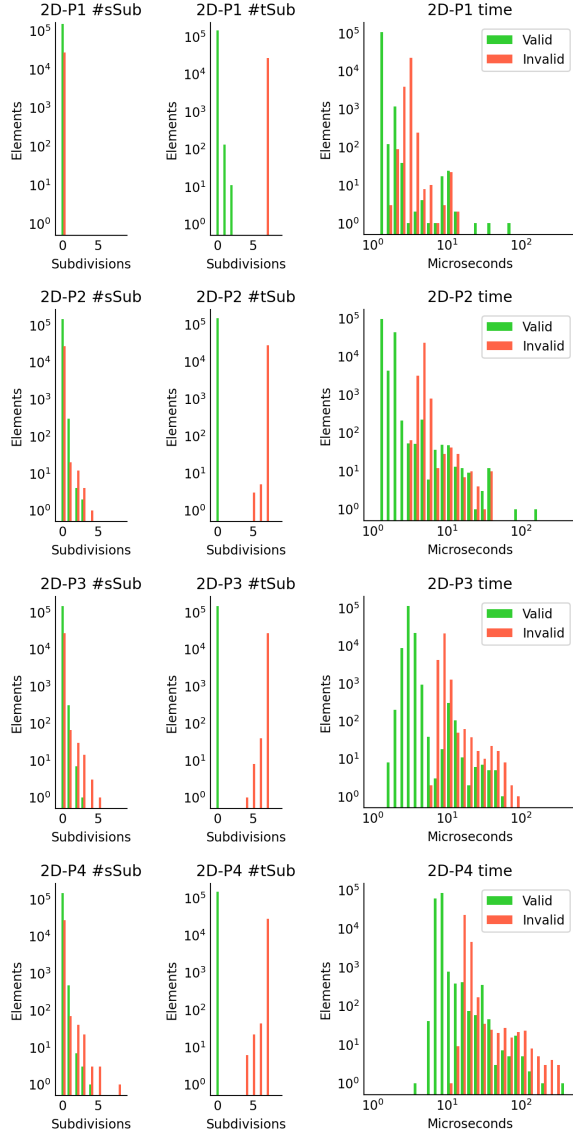
Fig. 10. Statistics for the Kangaroo datasets. Top to bottom: elements of order 1, 2, 3, 4; Left to right: number of space subdivisions (#ssub), time subdivisions (#tsub), and time to test an element. Green valid elements; red invalid ($t^* < 1$) elements.
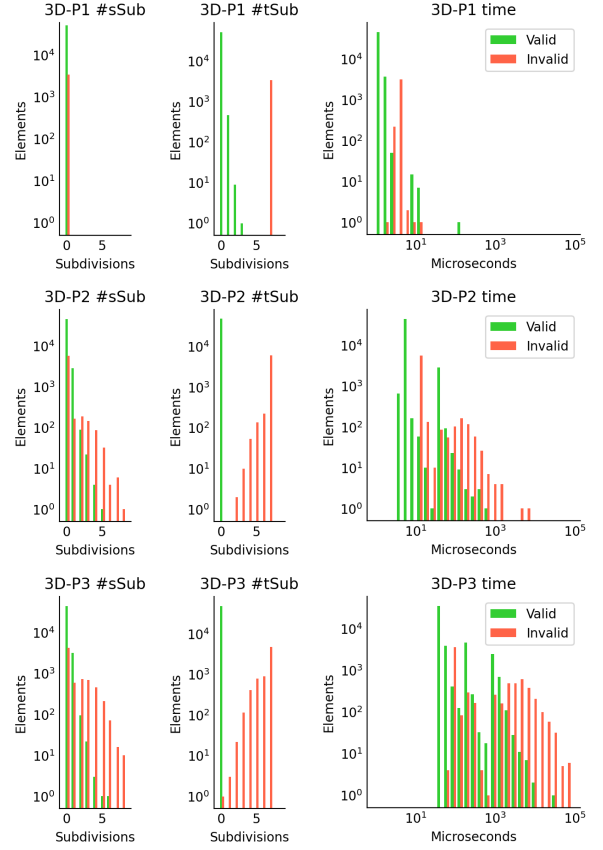

Fig. 11. Statistics for the Armadillo datasets. Top to bottom: elements of order 1, 2, 3; Left to right: number of space subdivisions (#ssub), time subdivisions (#tsub), and time to test an element. Green valid elements; red invalid ($t^* < 1$) elements.

9 CONCLUSIONS

We introduced a formulation for continuous inversion test and a corresponding conservative and efficient algorithm. Our solution addresses an open problem in existing finite element solver and parametrization algorithms, increasing robustness and providing, for the first time, a guarantee for interior point solvers to stay within the space of valid elements. While the issue does appear for linear elements, invalid elements are more commonly present in existing algorithms that use high-order bases. We believe our algorithm, and its reference implementation, will be a drop-in replacement for existing non-conservative checks used in many graphics algorithms that will increase robustness for a minor performance cost.

Limitations. The cost of our algorithm increases with the polynomial degree, especially for tensor product elements. While this is compatible with many applications in graphics and engineering (up to cubic tetrahedra, linear hexahedra/prisms), it is not scalable to very high-order FEM, where elements of order 20 or more are routinely used.

Additionally, similarly to all bisection algorithms, our approach might require a lot of refinement in certain queries where it is challenging to find the roots: while we could not find such a case for simplicial elements, our dataset contains some tensor product elements (57 hexahedra and 26 prisms) for which the algorithm exceeds the available memory on the personal machine where the benchmark was run, and must be stopped early via the l_{max} parameter (set to a maximum subdivision depth of 7): our algorithm still returns a conservative answer. Altering the subdivision strategy mitigates this issue and allows the algorithm to converge on the whole dataset: exploring better heuristics for subdivision may be a fruitful endeavor for improvement.
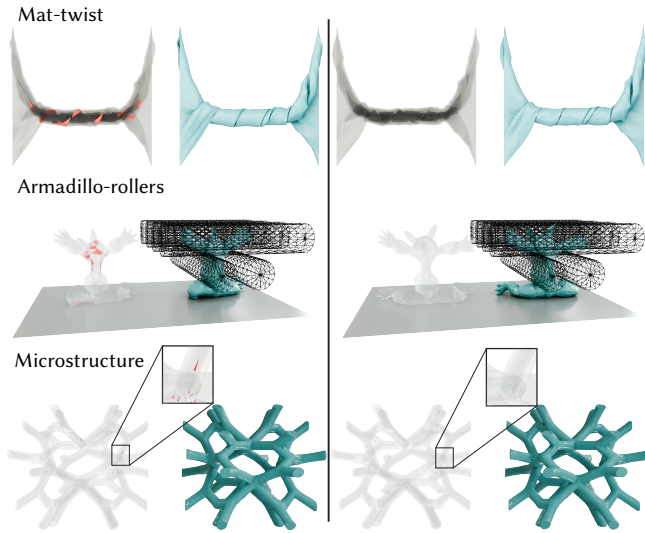
Fig. 12. Simulation examples in [Ferguson et al. 2023] with (right) and without (left) our inversion check. The high-order elements with non-positive Jacobian points are shown in red. On the left, the numbers of flipped elements are 20, 45, and 24 from top to bottom. Our method guarantees the positivity of Jacobian.
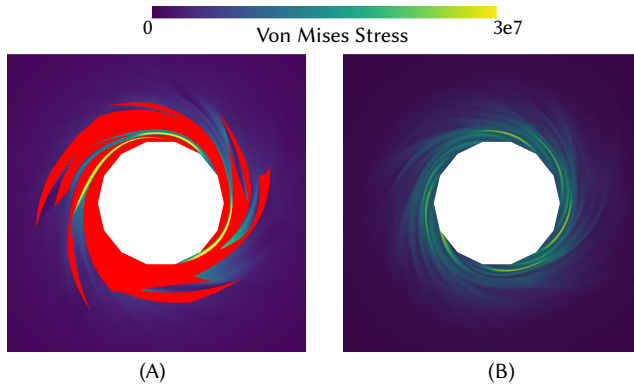

Fig. 13. Von Mises stress distribution of Ring-twist in Figure 1. The NaN is shown in red. (A) Without our method, NaN appears on elements with flipped points. (B) With our Jacobian check and adaptive quadrature, the stress is everywhere finite.

Future Work. Thanks to our approach, we discovered a previously unreported numerical problem with existing material models under extreme compression, which was obfuscated by inaccurate quadrature rules. We believe evaluating these material models in extreme compression regimes would be an interesting avenue for future work. Additionally, we discovered that high-order elastodynamic FE solvers often introduce inverted elements in their solutions: their effect on simulation accuracy requires further investigation, which is now possible since our test can detect them.

To further reduce the computational cost, it would be interesting to investigate the possibility of parallelizing individual queries or adapting our approach for massively parallel graphic processing units.

REFERENCES

Christie Alappat, Achim Basermann, Alan R. Bishop, Holger Fehske, Georg Hager, Olaf Schenk, Jonas Thies, and Gerhard Wellein. 2020. A Recursive Algebraic Coloring Technique for Hardware-Efficient Symmetric Sparse Matrix-Vector Multiplication. *ACM Transactions on Parallel Computing* 7, 3, Article 19 (June 2020), 37 pages.

R. W. Anderson, V. A. Dobrev, Tz. V. Kolev, and R. N. Rieben. 2014. Monotonicity in high-order curvilinear finite element arbitrary Lagrangian–Eulerian remap. *International Journal for Numerical Methods in Fluids* 77, 5 (Oct. 2014), 249–273. https://doi.org/10.1002/fld.3965

M. Attene. 2019. Indirect Predicates Library. https://github.com/MarcoAttene/Indirect_Predicates.

M. Attene. 2020. Indirect predicates for Geometric Constructions. *Computer-Aided Design* 126 (2020), 102856.

A. W. Bargteil and E. Cohen. 2014. Animation of Deformable Bodies with Quadratic Bézier Finite Elements. *ACM Trans. Graph.* 33, 3 (May 2014), 1–10.

Matthias Bollhöfer, Aryan Eftekhari, Simon Scheidegger, and Olaf Schenk. 2019. Large-scale Sparse Inverse Covariance Matrix Estimation. *SIAM Journal on Scientific Computing* 41, 1 (2019), 380–401.

Matthias Bollhöfer, Olaf Schenk, Radim Janalik, Steve Hamm, and Kiran Gullapalli. 2020. State-of-the-Art Sparse Direct Solvers. *Parallel Algorithms in Computational Science and Engineering* (2020), 3–33.

H. Brönnimann, C. Burnikel, and S. Pion. 1998. Interval Arithmetic Yields Efficient Dynamic Filters for Computational Geometry. In *Proc. 14th Symp. on Comp. Geom.* (Minneapolis, Minnesota, USA). ACM, New York, NY, USA, 165–174.

Yunuo Chen, Minchen Li, Lei Lan, Hao Su, Yin Yang, and Chenfanfu Jiang. 2022. A unified newton barrier method for multibody dynamics. *ACM Trans. Graph.* 41, 4, Article 66 (July 2022), 14 pages. https://doi.org/10.1145/3528223.3530076

J. A. Cottrell, T. J. R. Hughes, and Y. Bazilevs. 2009. *Isogeometric Analysis: Toward Integration of CAD and FEA*. Wiley.

Leonardo Dagum and Ramesh Menon. 1998. OpenMP: An Industry-Standard API for Shared-Memory Programming. *IEEE Comput. Sci. Eng.* 5, 1 (jan 1998), 46–55. https://doi.org/10.1109/99.660313

S. Dey, R.M. O'Bara, and M.S. Shephard. 2001. Towards curvilinear meshing in 3D: the case of quadratic simplices. *Computer-Aided Design* 33, 3 (2001), 199–209.

V. Dobrev, P. Knupp, T. Kolev, K. Mittal, and V. Tomov. 2019. The Target-Matrix Optimization Paradigm for High-Order Meshes. *SIAM Jou. Sci. Comp.* 41, 1 (2019), B50–B68.

Yu Fang, Minchen Li, Yadi Cao, Xuan Li, Joshuah Wolper, Yin Yang, and Chenfanfu Jiang. 2024. Augmented Incremental Potential Contact for Sticky Interactions. *IEEE Transactions on Visualization and Computer Graphics* 30, 8 (2024), 5596–5608. https://doi.org/10.1109/TVCG.2023.3295656

Yu Fang, Minchen Li, Chenfanfu Jiang, and Danny M. Kaufman. 2021. Guaranteed globally injective 3D deformation processing. *ACM Trans. Graph.* 40, 4, Article 75 (July 2021), 13 pages. https://doi.org/10.1145/3450626.3459757

G. Farin. 2001. *Curves and Surfaces for CAGD: A Practical Guide* (5th ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Zachary Ferguson et al. 2020. IPC Toolkit. https://ipc-sim.github.io/ipc-toolkit/. https://ipc-sim.github.io/ipc-toolkit/

Z. Ferguson, P. Jain, D. Zorin, T. Schneider, and D. Panozzo. 2023. High-Order Incremental Potential Contact for Elastodynamic Simulation on Curved Meshes. In *ACM SIGGRAPH 2023 Conference Proceedings* (, Los Angeles, CA, USA,) *(SIGGRAPH '23)*. Association for Computing Machinery, New York, NY, USA, Article 77, 11 pages.

Zachary Ferguson, Minchen Li, Teseo Schneider, Francisca Gil-Ureta, Timothy Langlois, Chenfanfu Jiang, Denis Zorin, Danny M. Kaufman, and Daniele Panozzo. 2021. Intersection-free rigid body dynamics. *ACM Trans. Graph.* 40, 4, Article 183 (July 2021), 16 pages. https://doi.org/10.1145/3450626.3459802

A. Gargallo-Peiró, X. Roca, J. Peraire, and J. Sarrate. 2015. Distortion and quality measures for validating and generating high-order tetrahedral meshes. *Engineering with Computers* 31, 3 (2015), 423–437.

P.L. George and H. Borouchaki. 2014. Validity of Lagrange (Bézier) and rational Bézier quads of degree 2. 99, 8 (2014), 611–632. https://doi.org/10.1002/nme.4696

C. Geuzaine, A. Johnen, J. Lambrechts, J. F. Remacle, and T. Toulorge. 2015. *The Generation of Valid Curvilinear Meshes*. Springer International Publishing, Cham, 15–39.

F. Goualard. 2005. Gaol: NOT Just Another Interval Library. https://sourceforge.net/projects/gaol/.

T. Granlund and the GMP development team. 2012. *GNU MP: The GNU Multiple Precision Arithmetic Library* (5.0.5 ed.). http://gmplib.org/.

Kemeng Huang, Floyd M. Chitalu, Huancheng Lin, and Taku Komura. 2024a. GIPC: Fast and Stable Gauss-Newton Optimization of IPC Barrier Energy. *ACM Trans. Graph.* 43, 2, Article 23 (March 2024), 18 pages. https://doi.org/10.1145/3643028

Zizhou Huang, Davi Colli Tozoni, Arvi Gjoka, Zachary Ferguson, Teseo Schneider, Daniele Panozzo, and Denis Zorin. 2024b. Differentiable solver for time-dependent deformation problems with contact. *ACM Trans. Graph.* 43, 3, Article 31 (May 2024), 30 pages. https://doi.org/10.1145/3657648

Z. Jiang, Z. Zhang, Y. Hu, T. Schneider, D. Zorin, and D. Panozzo. 2021a. Bijective and coarse high-order tetrahedral meshes. *ACM Trans. Graph.* 40, 4 (2021), 1–16.

Z. Jiang, Z. Zhang, Y. Hu, T. Schneider, D. Zorin, and D. Panozzo. 2021b. Bijective and coarse high-order tetrahedral meshes. *ACM Trans. Graph.* 40, 4 (2021), 1–16.

A. Johnen, C. Geuzaine, T. Toulorge, and J.-F. Remacle. 2018. Efficient computation of the minimum of shape quality measures on curvilinear finite elements. *Computer-Aided Design* 103 (2018), 24–33.

A. Johnen, J.-F. Remacle, and C. Geuzaine. 2013. Geometrical validity of curvilinear finite elements. *J. Comput. Phys.* 233 (2013), 359–372. https://doi.org/10.1016/j.jcp.2012.08.051

A. Johnen, J. F. Remacle, and C. Geuzaine. 2014. Geometrical Validity of High-Order Triangular Finite Elements. *Eng. with Comput.* 30, 3 (2014), 375–382.

A. Johnen, J.-C. Weill, and J.-F. Remacle. 2017. Robust and efficient validation of the linear hexahedral element. *Procedia Engineering* 203 (2017), 271–283.

C. Kane, J. E. Marsden, M. Ortiz, and M. West. 2000. Variational integrators and the Newmark algorithm for conservative and dissipative mechanical systems. *Internat. J. Numer. Methods Engrg.* 49, 10 (2000), 1295–1325.

O Knuppel. 1994. PROFIL/BIAS - A fast interval library. *Computing* 53, 3 (1994), 277–287. https://doi.org/10.1007/BF02307379

B. Lambov. 2008. Interval Arithmetic Using SSE-2. In *Reliable Implementation of Real Number Algorithms: Theory and Practice*, P. Hertling, C. M. Hoffmann, W. Luther, and N. Revol (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 102–113.

Lei Lan, Danny M. Kaufman, Minchen Li, Chenfanfu Jiang, and Yin Yang. 2022a. Affine body dynamics: fast, stable and intersection-free simulation of stiff materials. *ACM Trans. Graph.* 41, 4, Article 67 (July 2022), 14 pages. https://doi.org/10.1145/3528223.3530064

Lei Lan, Minchen Li, Chenfanfu Jiang, Huamin Wang, and Yin Yang. 2023. Second-order Stencil Descent for Interior-point Hyperelasticity. *ACM Trans. Graph.* 42, 4, Article 108 (July 2023), 16 pages. https://doi.org/10.1145/3592104

Lei Lan, Guanqun Ma, Yin Yang, Changxi Zheng, Minchen Li, and Chenfanfu Jiang. 2022b. Penetration-free projective dynamics on the GPU. *ACM Trans. Graph.* 41, 4, Article 69 (July 2022), 16 pages. https://doi.org/10.1145/3528223.3530069

Lei Lan, Yin Yang, Danny Kaufman, Junfeng Yao, Minchen Li, and Chenfanfu Jiang. 2021. Medial IPC: accelerated incremental potential contact with medial elastics. *ACM Trans. Graph.* 40, 4, Article 158 (July 2021), 16 pages. https://doi.org/10.1145/3450626.3459753

M. Lerch, G. Tischler, J.W.V. Gudenberg, W. Hofschuster, and W. Kramer. 2006. FILIB++, a Fast Interval Library Supporting Containment Computations. *ACM Trans. Math. Softw* 32, 2 (2006), 299–324.

M. Li, Z. Ferguson, T. Schneider, T. Langlois, D. Zorin, D. Panozzo, C. Jiang, and D. M. Kaufman. 2020. Incremental Potential Contact: Intersection- and Inversion-free Large Deformation Dynamics. *ACM Trans. Graph. (SIGGRAPH)* 39, 4, Article 49 (2020).

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. 2023b. Convergent Incremental Potential Contact. arXiv:2307.15908 [math.NA]

Minchen Li, Danny M. Kaufman, and Chenfanfu Jiang. 2021. Codimensional incremental potential contact. *ACM Trans. Graph.* 40, 4, Article 170 (July 2021), 24 pages. https://doi.org/10.1145/3450626.3459767

Xuan Li, Yu Fang, Lei Lan, Huamin Wang, Yin Yang, Minchen Li, and Chenfanfu Jiang. 2023a. Subspace-Preconditioned GPU Projective Dynamics with Contact for Cloth Simulation. In *SIGGRAPH Asia 2023 Conference Papers* (Sydney, NSW, Australia) *(SA '23)*. Association for Computing Machinery, New York, NY, USA, Article 1, 12 pages. https://doi.org/10.1145/3610548.3618157

Xuan Li, Yu Fang, Minchen Li, and Chenfanfu Jiang. 2022. BFEMP: Interpenetration-free MPM–FEM coupling with barrier contact. *Computer Methods in Applied Mechanics and Engineering* 390 (2 2022). https://doi.org/10.1016/j.cma.2021.114350

Xuan Li, Minchen Li, Xuchen Han, Huamin Wang, Yin Yang, and Chenfanfu Jiang. 2024. A Dynamic Duo of Finite Elements and Material Points. In *ACM SIGGRAPH 2024 Conference Papers* (Denver, CO, USA) *(SIGGRAPH '24)*. Association for Computing Machinery, New York, NY, USA, Article 97, 11 pages. https://doi.org/10.1145/3641519.3657449

S. A. Maas, B. J. Ellis, G. A. Ateshian, and J. A. Weiss. 2012. FEBio: Finite Elements for Biomechanics. *Jou. of Biomechanical Engineering* 134, 1 (2012).

M. Mandad and M. Campen. 2020. Efficient piecewise higher-order parametrization of discrete surfaces with local and global injectivity. *Computer-Aided Design* 127 (2020), 102862.

Z. Marschner, D. Palmer, P. Zhang, and J. Solomon. 2020. Hexahedral Mesh Repair via Sum-of-Squares Relaxation. *Computer Graphics Forum* 39, 5 (Aug. 2020), 133–147.

L. Martin, P. Jain, Z. Ferguson, T. Gholamalizadeh, F. Moshfeghifar, K. Erleben, D. Panozzo, S. Abramowitch, and T. Schneider. 2024. A systematic comparison between FEBio and PolyFEM for biomechanical systems. *Computer Methods and Programs in Biomedicine* 244 (2024), 107938.

Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. 2017. SymPy: symbolic computing in Python. *PeerJ Computer Science* 3 (Jan. 2017), e103. https://doi.org/10.7717/peerj-cs.103

J. Mezger, B. Thomaszewski, S. Pabst, and W. Straßer. 2009. Interactive physically-based shape editing. *Computer Aided Geometric Design* 26, 6 (Aug. 2009), 680–694.

Steven S Muchnick. 1997. *Advanced Compiler Design and Implementation*. Morgan Kaufmann, Oxford, England.

R.W. Ogden. 2013. *Non-Linear Elastic Deformations*. Dover Publications. https://books.google.com/books?id=52XDAgAAQBAJ

Julian Panetta, Qingnan Zhou, Luigi Malomo, Nico Pietroni, Paolo Cignoni, and Denis Zorin. 2015. Elastic textures for additive fabrication. *ACM Transactions on Graphics* 34, 4 (July 2015), 1–12. https://doi.org/10.1145/2766937

B. Schling. 2011. The Boost C++ Libraries. *XML Press* (2011).

T. Schneider, J. Dumas, X. Gao, D. Zorin, and D. Panozzo. 2019. PolyFEM. https://polyfem.github.io/.

T. Schneider, Y. Hu, X. Gao, J. Dumas, D. Zorin, and D. Panozzo. 2022. A Large-Scale Comparison of Tetrahedral and Hexahedral Elements for Solving Elliptic PDEs with the Finite Element Method. *ACM Trans. on Graph.* 41, 3 (2022), 1–14.

Xing Shen, Runyuan Cai, Mengxiao Bi, and Tangjie Lv. 2024. Preconditioned Nonlinear Conjugate Gradient Method for Real-time Interior-point Hyperelasticity. In *ACM SIGGRAPH 2024 Conference Papers* (Denver, CO, USA) *(SIGGRAPH '24)*. Association for Computing Machinery, New York, NY, USA, Article 96, 11 pages. https://doi.org/10.1145/3641519.3657490

J. R. Shewchuk. 1997. Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. *Discrete & Computational Geometry* 18, 3 (Oct. 1997), 305–363.

J. Smith and S. Schaefer. 2015. Bijective Parameterization with Free Boundaries. *ACM Trans. Graph.* 34, 4, Article 70 (2015), 9 pages.

J Snyder. 1992. Interval Analysis For Computer Graphics. *ACM SIGGRAPH* (1992), 121–130.

S. Suwelack, D. Lukarski, V. Heuveline, R. Dillmann, and S. Speidel. 2013. Accurate surface embedding for higher order finite elements. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '13)*. ACM.

X. Tang, Z. Ferguson, T. Schneider, D. Zorin, S. Kamil, and D. Panozzo. 2023. A Cross-Platform Benchmark for Interval Computation Libraries. In *Parallel Processing and Applied Mathematics*, R. Wyrzykowski, J. Dongarra, E. Deelman, and K. Karczewski (Eds.). Springer International Publishing, Cham, 415–427.

Thomas Toulorge, Christophe Geuzaine, Jean-François Remacle, and Jonathan Lambrechts. 2013. Robust untangling of curvilinear meshes. *J. Comput. Phys.* 254 (Dec. 2013), 8–26. https://doi.org/10.1016/j.jcp.2013.07.022

O. V. Ushakova. 2011. Nondegeneracy tests for hexahedral cells. *Computer Methods in Applied Mechanics and Engineering* 200, 17–20 (2011), 1649–1658.

S. Vavasis. 2003. A Bernstein-Bezier Sufficient Condition for Invertibility of Polynomial Mapping Functions. (2003).

Bolun Wang, Zachary Ferguson, Teseo Schneider, Xin Jiang, Marco Attene, and Daniele Panozzo. 2021a. A Large-scale Benchmark and an Inclusion-based Algorithm for Continuous Collision Detection. *ACM Trans. Graph.* 40, 5, Article 188 (sep 2021), 16 pages. https://doi.org/10.1145/3460775

B. Wang, Z. Ferguson, T. Schneider, X. Jiang, M. Attene, and D. Panozzo. 2021b. A Large-scale Benchmark and an Inclusion-based Algorithm for Continuous Collision Detection. *ACM Trans. Graph.* 40, 5 (2021), 1–16.

Inc. Wolfram Research. 2023. Mathematica, Version 13.3. https://www.wolfram.com/mathematica Champaign, IL.

A LANGRANGE AND BÉZIER FORMS OF $|J_x|$

Let $f : \sigma_s^n \longrightarrow \mathbb{R}$ be a polynomial of order p defined on a standard element according to Definition 1. We detail the representation of f in Lagrange and Bézier forms and the conversions between these two representations.

Lagrange form. Let $\Gamma_\sigma^p = (1/p)\mathbb{Z}^n \cap \sigma$ be a grid of uniformly distributed domain points of σ, and $\mathcal{I}_\sigma^p$ be its set of indices (Figure 5). The Lagrange basis of order p consists of $|\Gamma_\sigma^p|$ order p polynomials such that for each point $\gamma_i \in \Gamma_\sigma^p$, $\mathcal{L}_i^p(\gamma_j) = \delta_{ij}$. A function f is represented in the Lagrange basis as:

$$f(\xi) = \sum_{i \in \mathcal{I}_\sigma^p} y_i \mathcal{L}_i^p(\xi), \tag{6}$$

where $y_i = f(\gamma_i)$ for $i \in \mathcal{I}_\sigma^p$.

Bézier form. aThe same function f can be expressed equivalently in Bézier form by using a Bernstein basis on the same set Γ_σ^p of domain points:

$$f(\xi) = \sum_{i \in \mathcal{I}_\sigma^p} \beta_i \mathcal{B}_i^p(\xi), \tag{7}$$

where the $\mathcal{B}_i^p$ are Bernstein polynomials of order p and the β_i are their corresponding control coefficients. Unlike the Lagrange case, β_i equals $f(\gamma_i)$ only at the corners of σ. However, the graph of f is contained in the convex hull of points (γ_i, β_i), providing a simple way to bound f from below and above at all points of σ [Farin 2001].

Transformation matrix. Let us denote $f^{\mathcal{L}}$ the vector consisting of all the y_i, and likewise, $f^{\mathcal{B}}$ the vector consisting of all the β_i, for $i \in \mathcal{I}_\sigma^p$. We can convert between the two representations through transformation matrices [Johnen et al. 2014]:

$$f^{\mathcal{L}} = \mathbf{T}_{\mathcal{B} \to \mathcal{L}} f^{\mathcal{B}} \qquad f^{\mathcal{B}} = \mathbf{T}_{\mathcal{L} \to \mathcal{B}} f^{\mathcal{L}}. \tag{8}$$

Such matrices depend only on the reference element σ_s^n and the order p thus can be computed once for each element type and order. Matrix $\mathbf{T}_{\mathcal{B} \to \mathcal{L}}$ is easily computed by evaluation of Bernstein polynomials on Γ_σ^p:

$$(\mathbf{T}_{\mathcal{B} \to \mathcal{L}})_{ij} = \mathcal{B}_j^p(\gamma_i) \quad \forall i, j \in \mathcal{I}_\sigma^p \tag{9}$$

and $\mathbf{T}_{\mathcal{L} \to \mathcal{B}}$ is the inverse of $\mathbf{T}_{\mathcal{B} \to \mathcal{L}}$.

Subdivision matrices. For each q, we first build a transformation matrix from the Bézier basis to the Lagrange basis that interpolates the domain points $\psi^q(\Gamma(m))$ of the q-th subdomain. This is analogous to building matrix $\mathbf{T}_{\mathcal{B} \to \mathcal{L}}$, by sampling the Bézier basis on $\psi^q(\Gamma(m))$ instead of $\Gamma(m)$:

$$(\mathbf{T}_{\mathcal{B} \to \mathcal{L}}^q)_{ij} = \mathcal{B}_j^m(\psi^q(\gamma_i)) \quad \forall i, j \in \mathcal{I}_\sigma(m). \tag{10}$$

Then we multiply it with the Lagrange-to-Bézier matrix to build

$$\mathbf{T}_{\mathcal{B} \to \mathcal{B}}^q = \mathbf{T}_{\mathcal{L} \to \mathcal{B}} \mathbf{T}_{\mathcal{B} \to \mathcal{L}}^q, \tag{11}$$

which allows us to go directly from the Bézier coefficients on the domain to the Bézier coefficients on each subdomain.

Time subdivision matrices. Time subdivision matrices defined by plugging the time subdivision maps

$$\psi^-(\xi, t) = (\xi, t/2), \qquad \psi^+(\xi, t) = (\xi, (t+1)/2) \tag{12}$$

in Equation (10) and using Equation (11) as above.

Representations of the Jacobian determinant. Given a standard element σ_s^n as above, let us define $m_\triangle = np - s$ and $m_\square = np - 1$. Let us denote $i_0 = m_\triangle - (\sum_{j=1}^s i_j)$ and $\xi_0 = 1 - (\sum_{j=1}^s \xi_j)$.

The Lagrange basis polynomials used to represent the Jacobian determinant $|J_x(\xi)|$ on reference element $\sigma = \sigma_s^n$ are:

$$\mathcal{L}_{i_1,\ldots,i_n}^\sigma(\xi_1,\ldots,\xi_n) = \left(\prod_{j=0}^s \ell_{i_j}^{i_j,m_\triangle}(\xi_{i_j})\right)\left(\prod_{j=s+1}^n \ell_{i_j}^{m_\square,m_\square}(\xi_{i_j})\right) \tag{13}$$

$$\ell_j^{q,m}(\zeta) = \prod_{k \in \{0,\ldots,q\} \setminus \{j\}} \frac{m\zeta - k}{j - k} \tag{14}$$

The Bézier basis polynomials used to represent the Jacobian determinant $|J_x(\xi)|$ on reference element $\sigma = \sigma_s^n$ are:

$$\mathcal{B}_{i_1,\ldots,i_n}^\sigma(\xi_1,\ldots,\xi_n) = \left(\binom{m_\triangle}{i_0,\ldots,i_s}\xi_0^{i_0}\ldots\xi_s^{i_s}\right)\prod_{j=s+1}^n b_{i_j}^{m_\square}(\xi_{i_j}) \tag{15}$$

$$b_j^q(\zeta) = \binom{q}{j}\zeta^j(1-\zeta)^{q-j} \tag{16}$$

where $\mathcal{B}_{i_1,\ldots,i_n}^\sigma$ is the product of an order $m_\triangle$ Bernstein polynomial on the s-simplex basis, in the variables $\xi_1,\ldots,\xi_s$, with an order $m_\square$ Bernstein polynomial on the $(n-s)$-tensor product basis in the variables $\xi_{s+1},\ldots,\xi_n$, which is itself a product of $(n-s)$ univariate Bernstein polynomials of order $m_\square$. It is easy to see that

- the order of $|J_x(\xi)|$ in $\{\xi_1,\ldots,\xi_s\}$ is $m_\triangle = np - s$;
- the order of $|J_x(\xi)|$ in $\{\xi_{s+1},\ldots,\xi_n\}$ is $m_\square = np - 1$.

Notice that on n-simplices ($s = n$) and n-hypercubes ($s = 1$), $|J_x|$ has the same order in every variable.

Extension to Dynamic. For the dynamic case, we add a term for time to the Bézier basis construction, an order n univariate Bernstein polynomial in t:

$$\bar{\mathcal{B}}_{i_1,\ldots,i_n}^\sigma(\xi_1,\ldots,\xi_n,t) = \mathcal{B}_{i_1,\ldots,i_n,i_t}^\sigma(\xi_1,\ldots,\xi_n)b_{i_t}^n(t) \tag{17}$$

It is easy to see that the order of $|J_{\bar{x}}|$ remains the same as the static case in its spatial variables, while it is n in its time variable. The number of polynomials in the Bernstein basis, which is equal to the number of terms of the given polynomial, increases combinatorially with the dimension n and order p and is given by

$$N = \binom{np}{s}(np)^{n-s}, \qquad \bar{N} = N(n+1), \tag{18}$$

for the static and the dynamic case, respectively. Note that, this combinatorial growth poses intrinsic limitations to scaling the problem up in degree and dimension. Nevertheless, it remains tractable for the most common cases, as exemplified in Table 5.

B SUBDIVISION RULES

The subdivision functions ψ^q for the various types of elements are listed below. Their related subdomains are shown in the insets (red bullet = origin). The dynamic elements for $n = 3$ require 4D hypercubes (for tensor product) and hyper-prisms (for simplicial and mixed).

Element name	n	s	p	$\xi_{1\ldots s}$	$\xi_{s+1\ldots n}$	t	$\bar{N}$
Linear triangle	2	2	1	0	-	2	3
Quadratic triangle	2	2	2	2	-	2	18
Cubic triangle	2	2	3	4	-	2	45
Quartic triangle	2	2	4	6	-	2	84
Quintic triangle	2	2	5	8	-	2	135
Bilinear quadrangle	2	1	1	1	1	2	12
Biquadratic quadrangle	2	1	2	3	3	2	48
Bicubic quadrangle	2	1	3	5	5	2	108
Linear tetrahedron	3	3	1	0	-	3	4
Quadratic tetrahedron	3	3	2	3	-	3	80
Cubic tetrahedron	3	3	3	6	-	3	336
Quartic tetrahedron	3	3	4	9	-	3	880
Bilinear tri. prism	3	2	1	1	2	3	36
Biquadratic tri. prism	3	2	2	4	5	3	360
Trilinear hexahedron	3	1	1	2	2	3	108
Triquadratic hexahedron	3	1	2	5	5	3	864

Table 5. Order of $|J_x(\xi)|$ in the spatial variables ξ and the time variable t, and the number $\bar{N}$ of its terms for a generic order p map on $\bar{\sigma}_s^n$.
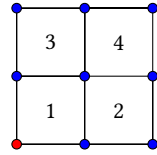
Triangle.

$$\psi^1(\xi) = (\xi_1/2, \xi_2/2)$$
$$\psi^2(\xi) = ((\xi_1+1)/2, \xi_2/2)$$
$$\psi^3(\xi) = (\xi_1/2, (\xi_2+1)/2)$$
$$\psi^4(\xi) = ((1-\xi_1)/2, (1-\xi_2)/2)$$
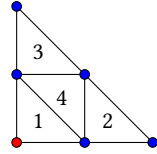
Quad.

$$\psi^1(\xi) = (\xi_1/2, \xi_2/2)$$
$$\psi^2(\xi) = ((\xi_1+1)/2, \xi_2/2)$$
$$\psi^3(\xi) = (\xi_1/2, (\xi_2+1)/2)$$
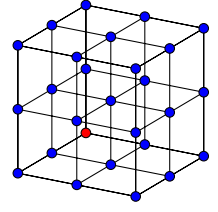$$\psi^4(\xi) = ((\xi_1+1)/2, (\xi_2+1)/2)$$

Tetrahedron. The subdivision of the tetrahedron is non-trivial: it is split into four tetrahedra incident at the corners of the domain (corresponding to subdomains 1–4) and a central octahedron, which is further split into four tetrahedra (domains 5–8).

$$\psi^1(\xi) = (\xi_1/2, \xi_2/2, \xi_3/2)$$
$$\psi^2(\xi) = ((\xi_1+1)/2, \xi_2/2, \xi_3/2)$$
$$\psi^3(\xi) = (\xi_1/2, (\xi_2+1)/2, \xi_3/2)$$
$$\psi^4(\xi) = (\xi_1/2, \xi_2/2, (\xi_3+1)/2)$$
$$\psi^5(\xi) = ((1-\xi_2-\xi_3)/2, \xi_2/2, (\xi_1+\xi_2+\xi_3)/2)$$
$$\psi^6(\xi) = ((1-\xi_2)/2, (\xi_1+\xi_2)/2, (\xi_2+\xi_3)/2)$$
$$\psi^7(\xi) = ((\xi_1+\xi_2)/2, (1-\xi_1)/2, \xi_3/2)$$
$$\psi^8(\xi) = (\xi_1/2, (\xi_2+\xi_3)/2, (1-\xi_1+\xi_2)/2)$$
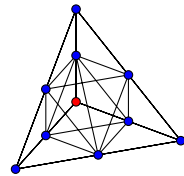
Hexa. The subdivision of a hexahedron works both for a static hexahedral domain and a dynamic quad domain. In the latter case, the third coordinate is time.

$$\psi^1(\xi) = (\xi_1/2, \xi_2/2, \xi_3/2)$$
$$\psi^2(\xi) = ((\xi_1+1)/2, \xi_2/2, \xi_3/2)$$
$$\psi^3(\xi) = (\xi_1/2, (\xi_2+1)/2, \xi_3/2)$$
$$\psi^4(\xi) = ((\xi_1+1)/2, (\xi_2+1)/2, \xi_3/2)$$
$$\psi^5(\xi) = (\xi_1/2, \xi_2/2, (\xi_3+1)/2)$$
$$\psi^6(\xi) = ((\xi_1+1)/2, \xi_2/2, (\xi_3+1)/2)$$
$$\psi^7(\xi) = (\xi_1/2, (\xi_2+1)/2, (\xi_3+1)/2)$$
$$\psi^8(\xi) = ((\xi_1+1)/2, (\xi_2+1)/2, (\xi_3+1)/2)$$

Prism. The subdivision of a prism works both for a static prism domain and a dynamic triangle domain. In the latter case, the third coordinate is time.

$$\psi^1(\xi) = (\xi_1/2, \xi_2/2, \xi_3/2)$$
$$\psi^2(\xi) = ((\xi_1+1)/2, \xi_2/2, \xi_3/2)$$
$$\psi^3(\xi) = (\xi_1/2, (\xi_2+1)/2, \xi_3/2)$$
$$\psi^4(\xi) = ((1-\xi_1)/2, (1-\xi_2)/2, \xi_3/2)$$
$$\psi^5(\xi) = (\xi_1/2, \xi_2/2, (\xi_3+1)/2)$$
$$\psi^6(\xi) = ((\xi_1+1)/2, \xi_2/2, (\xi_3+1)/2)$$
$$\psi^7(\xi) = (\xi_1/2, (\xi_2+1)/2, (\xi_3+1)/2)$$
$$\psi^8(\xi) = ((1-\xi_1)/2, (1-\xi_2)/2, (\xi_3+1)/2)$$
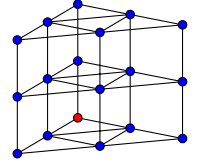
Hypercube. This is a 4-hypercube that works as a domain for a dynamic hexahedral element.

$$\psi^1(\xi,t) = (\xi_1/2, \xi_2/2, \xi_3/2, t/2, t/2)$$
$$\psi^2(\xi,t) = ((\xi_1+1)/2, \xi_2/2, \xi_3/2, t/2, t/2)$$
$$\psi^3(\xi,t) = (\xi_1/2, (\xi_2+1)/2, \xi_3/2, t/2)$$
$$\psi^4(\xi,t) = ((\xi_1+1)/2, (\xi_2+1)/2, \xi_3/2, t/2)$$
$$\psi^5(\xi,t) = (\xi_1/2, \xi_2/2, (\xi_3+1)/2, t/2)$$
$$\psi^6(\xi,t) = ((\xi_1+1)/2, \xi_2/2, (\xi_3+1)/2, t/2)$$
$$\psi^7(\xi,t) = (\xi_1/2, (\xi_2+1)/2, (\xi_3+1)/2, t/2)$$
$$\psi^8(\xi,t) = ((\xi_1+1)/2, (\xi_2+1)/2, (\xi_3+1)/2, t/2)$$
$$\psi^9(\xi,t) = (\xi_1/2, \xi_2/2, \xi_3/2, t/2, (t+1)/2)$$
$$\psi^{10}(\xi,t) = ((\xi_1+1)/2, \xi_2/2, \xi_3/2, t/2, (t+1)/2)$$
$$\psi^{11}(\xi,t) = (\xi_1/2, (\xi_2+1)/2, \xi_3/2, (t+1)/2)$$
$$\psi^{12}(\xi,t) = ((\xi_1+1)/2, (\xi_2+1)/2, \xi_3/2, (t+1)/2)$$
$$\psi^{13}(\xi,t) = (\xi_1/2, \xi_2/2, (\xi_3+1)/2, (t+1)/2)$$
$$\psi^{14}(\xi,t) = ((\xi_1+1)/2, \xi_2/2, (\xi_3+1)/2, (t+1)/2)$$
$$\psi^{15}(\xi,t) = (\xi_1/2, (\xi_2+1)/2, (\xi_3+1)/2, (t+1)/2)$$
$$\psi^{16}(\xi,t) = ((\xi_1+1)/2, (\xi_2+1)/2, (\xi_3+1)/2, (t+1)/2)$$

Hyperprism 1. This is the tensor product of a tetrahedron with an interval that works as a domain for a dynamic tetrahedral element.

$$\psi^1(\xi, t) = (\xi_1/2, \xi_2/2, \xi_3/2, t/2)$$

$$\psi^2(\xi, t) = ((\xi_1 + 1)/2, \xi_2/2, \xi_3/2, t/2)$$

$$\psi^3(\xi, t) = (\xi_1/2, (\xi_2 + 1)/2, \xi_3/2, t/2)$$

$$\psi^4(\xi, t) = (\xi_1/2, \xi_2/2, (\xi_3 + 1)/2, t/2)$$

$$\psi^5(\xi, t) = ((1 - \xi_2 - \xi_3)/2, \xi_2/2, (\xi_1 + \xi_2 + \xi_3)/2, t/2)$$

$$\psi^6(\xi, t) = ((1 - \xi_2)/2, (\xi_1 + \xi_2)/2, (\xi_2 + \xi_3)/2, t/2)$$

$$\psi^7(\xi, t) = ((\xi_1 + \xi_2)/2, (1 - \xi_1)/2, \xi_3/2, t/2)$$

$$\psi^8(\xi, t) = (\xi_1/2, (\xi_2 + \xi_3)/2, (1 - \xi_1 + \xi_2)/2, t/2)$$

$$\psi^9(\xi, t) = (\xi_1/2, \xi_2/2, \xi_3/2, (t + 1)/2)$$

$$\psi^{10}(\xi, t) = ((\xi_1 + 1)/2, \xi_2/2, \xi_3/2, (t + 1)/2)$$

$$\psi^{11}(\xi, t) = (\xi_1/2, (\xi_2 + 1)/2, \xi_3/2, (t + 1)/2)$$

$$\psi^{12}(\xi, t) = (\xi_1/2, \xi_2/2, (\xi_3 + 1)/2, (t + 1)/2)$$

$$\psi^{13}(\xi, t) = ((1 - \xi_2 - \xi_3)/2, \xi_2/2, (\xi_1 + \xi_2 + \xi_3)/2, (t + 1)/2)$$

$$\psi^{14}(\xi, t) = ((1 - \xi_2)/2, (\xi_1 + \xi_2)/2, (\xi_2 + \xi_3)/2, (t + 1)/2)$$

$$\psi^{15}(\xi, t) = ((\xi_1 + \xi_2)/2, (1 - \xi_1)/2, \xi_3/2, (t + 1)/2)$$

$$\psi^{16}(\xi, t) = (\xi_1/2, (\xi_2 + \xi_3)/2, (1 - \xi_1 + \xi_2)/2, (t + 1)/2)$$

Hyperprism 2. This is the tensor product of a prism with an interval that works as a domain for a dynamic prism element.

$$\psi^1(\xi.t) = (\xi_1/2, \xi_2/2, \xi_3/2, t/2)$$

$$\psi^2(\xi.t) = ((\xi_1 + 1)/2, \xi_2/2, \xi_3/2, t/2)$$

$$\psi^3(\xi.t) = (\xi_1/2, (\xi_2 + 1)/2, \xi_3/2, t/2)$$

$$\psi^4(\xi.t) = ((1 - \xi_1)/2, (1 - \xi_2)/2, \xi_3/2, t/2)$$

$$\psi^5(\xi.t) = (\xi_1/2, \xi_2/2, (\xi_3 + 1)/2, t/2)$$

$$\psi^6(\xi.t) = ((\xi_1 + 1)/2, \xi_2/2, (\xi_3 + 1)/2, t/2)$$

$$\psi^7(\xi.t) = (\xi_1/2, (\xi_2 + 1)/2, (\xi_3 + 1)/2, t/2)$$

$$\psi^8(\xi.t) = ((1 - \xi_1)/2, (1 - \xi_2)/2, (\xi_3 + 1)/2, t/2)$$

$$\psi^9(\xi.t) = (\xi_1/2, \xi_2/2, \xi_3/2, (t + 1)/2)$$

$$\psi^{10}(\xi.t) = ((\xi_1 + 1)/2, \xi_2/2, \xi_3/2, (t + 1)/2)$$

$$\psi^{11}(\xi.t) = (\xi_1/2, (\xi_2 + 1)/2, \xi_3/2, (t + 1)/2)$$

$$\psi^{12}(\xi.t) = ((1 - \xi_1)/2, (1 - \xi_2)/2, \xi_3/2, (t + 1)/2)$$

$$\psi^{13}(\xi.t) = (\xi_1/2, \xi_2/2, (\xi_3 + 1)/2, (t + 1)/2)$$

$$\psi^{14}(\xi.t) = ((\xi_1 + 1)/2, \xi_2/2, (\xi_3 + 1)/2, (t + 1)/2)$$

$$\psi^{15}(\xi.t) = (\xi_1/2, (\xi_2 + 1)/2, (\xi_3 + 1)/2, (t + 1)/2)$$

$$\psi^{16}(\xi.t) = ((1 - \xi_1)/2, (1 - \xi_2)/2, (\xi_3 + 1)/2, (t + 1)/2)$$

C INTERVAL ARITHMETIC

The interval type we use defines the following operations and relations:

$$-[\underline{x}, \overline{x}] = [-\overline{x}, -\underline{x}]$$

$$[\underline{x}, \overline{x}] + [\underline{y}, \overline{y}] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$$

$$[\underline{x}, \overline{x}][\underline{y}, \overline{y}] = [\min(\underline{x}\,\underline{y}, \underline{x}\,\overline{y}, \overline{x}\,\underline{y}, \overline{x}\,\overline{y}), \max(\underline{x}\,\underline{y}, \underline{x}\,\overline{y}, \overline{x}\,\underline{y}, \overline{x}\,\overline{y})]$$

$$\min([\underline{x}, \overline{x}], [\underline{y}, \overline{y}]) = [\min(\underline{x}, \underline{y}), \min(\overline{x}, \overline{y})]$$

$$\max([\underline{x}, \overline{x}], [\underline{y}, \overline{y}]) = [\max(\underline{x}, \underline{y}), \max(\overline{x}, \overline{y})]$$

$$[\underline{x}, \overline{x}] = [\underline{y}, \overline{y}] \Leftrightarrow \underline{x} = \underline{y} \wedge \overline{x} = \overline{y}$$

$$[\underline{x}, \overline{x}] > [\underline{y}, \overline{y}] \Leftrightarrow \underline{x} > \overline{y}$$

The division between intervals should be avoided, as the divisor may contain 0. However, a division of an interval by an exact number is acceptable; in our code, we have only divisions by 2, which is however exact even in floating point.

Note that there is no total ordering of intervals, so for example, $[\underline{x}, \overline{x}] \not\leq 0 \Rightarrow [\underline{x}, \overline{x}] > 0$.

In our implementation, outward rounding is achieved by internally storing the lower end of the interval with the opposite sign and changing the processor rounding mode to always rounding up. This ensures that the interval's width increases only when the actual floating-point computation is inexact.

D GROUND TRUTH

We designed a Mathematica script that computes symbolically the roots of $|J_{\hat{x}}|$ for a given element. The script receives in input the type, dimension n, and order p of an element together with the control points describing its geometric map at times $t = 0$ and $t = 1$. This is the same input taken by our test. The script computes the symbolic expression of $|J_{\hat{x}}|$, which is a polynomial in $n + 1$ variables, finds its roots inside the domain $\hat{\sigma}$, and returns the minimum value of t in which a root is found. First, we use `FindInstance` to check if the region where $J \leq 0$ is empty. If so, the element is valid throughout; otherwise, we use `Minimize` to find the minimum time t in such a region. The scripts will be released as part of our open-source code.

E FAILURE CASE FOR FLOATING POINT ALGORITHM

We present an artificial example where floating point error causes an invalid element to be detected as valid by the non-robust implementation of the static validity check. Consider the static quadratic triangle with control points:

$$\mathbf{v}_{00} = (0.0, -1.0)$$

$$\mathbf{v}_{20} = (0.9999999999999997, -1.0)$$

$$\mathbf{v}_{02} = (-1.1093356479670479e{-}16, -3.3306690738754696e{-}16)$$

$$\mathbf{v}_{01} = (0.5, -0.75)$$

$$\mathbf{v}_{11} = (0.75, -0.25)$$

$$\mathbf{v}_{10} = (0.25000000000000017, -0.5)$$

control points $\mathbf{v}_{00}$, $\mathbf{v}_{20}$, and $\mathbf{v}_{02}$ are the three vertices of the triangle, and $\mathbf{v}_{01}$, $\mathbf{v}_{11}$, $\mathbf{v}_{10}$ are the edge midpoints. The value of the Jacobian determinant is positive at all control points, except $\mathbf{v}_{00}$. At this point, the Jacobian is

$$
\begin{aligned}
J_{00} =& (4\mathbf{v}_{01}^x - 3\mathbf{v}_{00}^x - \mathbf{v}_{20}^x)(4\mathbf{v}_{10}^y - 3\mathbf{v}_{00}^y - \mathbf{v}_{02}^y) - \\
& (4\mathbf{v}_{10}^x - 3\mathbf{v}_{00}^x - \mathbf{v}_{02}^x)(4\mathbf{v}_{01}^y - 3\mathbf{v}_{00}^y - \mathbf{v}_{20}^y) \\
=& -1.1093356479670467e{-}16 < 0
\end{aligned}
$$

when computed with exact rational arithmetic. However, when tested with the floating point implementation, the Jacobian determinant at this vertex evaluates to $2.220446049250313e{-}16 > 0$; because all other Bézier coefficients are positive as well, the element is classified as valid.

Our conservative implementation takes representation error into account and does not converge for this element, meaning that the algorithm will stop after $l_{\max}$ subdivisions leaving the element undecided, and thus it will be treated as invalid.