Poly-Spline Finite-Element Method

TESEO SCHNEIDER, New York University
JÉRÉMIE DUMAS, New York University, nTopology
XIFENG GAO, New York University, Florida State University
MARIO BOTSCH, Bielefeld University
DANIELE PANOZZO and DENIS ZORIN, New York University

Fig. 1. A selection of the automatically generated pure hexahedral and hexahedral-dominant meshes in our test set. The colors denote the type of basis used. In the bottom-right, we show the result of a Poisson problem solved over a hex-dominant, polyhedral mesh.

We introduce an integrated meshing and finite-element method pipeline enabling solution of partial differential equations in the volume enclosed by a boundary representation. We construct a hybrid hexahedral-dominant mesh, which contains a small number of star-shaped polyhedra, and build a set of high-order bases on its elements, combining triquadratic B-splines, triquadratic hexahedra, and harmonic elements. We demonstrate that our approach converges cubically under refinement, while requiring around 50% of the degrees of freedom than a similarly dense hexahedral mesh composed of triquadratic hexahedra. We validate our approach solving Poisson's equation on a large collection of models, which are automatically processed by our algorithm, only requiring the user to provide boundary conditions on their surface.

We are grateful to the NYU HPC staff for providing computing cluster service. This work was partially supported by the NSF CAREER award 1652515, the NSF grant IIS-1320635, the NSF grant DMS-1436591, the NSF grant 1835712, the SNSF grant P2TIP2_175859, a gift from Adobe Research, and a gift from nTopology.
Authors' addresses: T. Schneide, D. Panozzo, and D. Zorin, NYU Courant Institute of Mathematical Sciences, 60 5th Ave, New York, NY 10011; emails: {teseo.schneider, panozzo}@nyu.edu, dzorin@cs.nyu.edu; J. Dumas, nTopology, 153 Lafayette St, New York, NY 10013; email: jeremie.dumas@ens-lyon.org; X. Gao, Computer Science Department, Florida State University, 171 James Love Building, 1017 Academic Way, Tallahassee, FL 32306; email: gxf.xisha@gmail.com; M. Botsch, Bielefeld University, Inspiration 1, 33615 Bielefeld, Germany; email: botsch@techfak.uni-bielefeld.de.

0730-0301/2019/03-ART19 $15.00
https://doi.org/10.1145/3313797

CCS Concepts: • **Computing methodologies** → **Modeling and simulation**; **Physical simulation**; *Mesh geometry models*; • **Mathematics of computing** → *Mesh generation*;

Additional Key Words and Phrases: Finite elements, polyhedral meshes, splines, simulation

ACM Reference format:
Teseo Schneider, Jérémie Dumas, Xifeng Gao, Mario Botsch, Daniele Panozzo, and Denis Zorin. 2019. Poly-Spline Finite-Element Method. *ACM Trans. Graph.* 38, 3, Article 19 (March 2019), 16 pages.
https://doi.org/10.1145/3313797

1 INTRODUCTION

The numerical solution of partial differential equations is ubiquitous in computer graphics and engineering applications, ranging from the computation of UV maps and skinning weights to the simulation of elastic deformations, fluids, and light scattering.

The finite-element method (FEM) is the most commonly used discretization of partial differential equations (PDEs), especially in the context of structural and thermal analysis, due to its generality and rich selection of off-the-shelf commercial implementations. Ideally, a PDE solver should be a "black box": The user provides as input the domain boundary, boundary conditions, and the governing equations, and the code returns an evaluator that can compute the value of the solution at any point of the input domain. This is surprisingly far from being the case for all existing open-source or

commercial software, despite the research efforts in this direction and the large academic and industrial interest.

To a large extent, this is due to treating meshing and FEM basis construction as two disjoint problems. The FEM basis construction may make a seemingly innocuous assumption (e.g., on the geometry of elements) that leads to exceedingly difficult requirements for meshing software. For example, commonly used bases for tetrahedra are sensitive to the tetrahedron shape, so tetrahedral mesh generators have to guarantee good element shape everywhere: a difficult task that, for some surfaces, does not have a fully satisfactory solution. Alternatively, if few assumptions are made on mesh generation (e.g., one can use elements that work on arbitrary polyhedral domains), then the basis and stiffness matrix constructions can become very expensive.

This state of matters presents a fundamental problem for applications that require fully automatic, robust processing of large collections of meshes of varying sizes, an increasingly common situation as large collections of geometric data become available. Most importantly, this situation arises in the context of machine learning on geometric and physical data, where a neural network could be trained using large numbers of simulations and used to compute efficiently an approximated solution (Chen et al. 2018; Kostrikov et al. 2018). Similarly, shape optimization problems often require solving PDEs in the inner optimization loop on a constantly changing domain (Panetta et al. 2015).

Overview. We propose an integrated pipeline, considering meshing and element design as a single challenge: We make the tradeoff between mesh quality and element complexity/cost *local*, instead of making an *a priori* decision for the whole pipeline. We generate high-quality, simple, and regularly arranged elements for most of the volume of the shape, with more complex and poor quality polyhedral shapes filling the remaining gaps (Gao et al. 2017a; Sokolov et al. 2016). Our idea is to match each element to a basis construction, with well-shaped elements getting the simplest and most efficient basis functions and with complex polyhedral element formulations used only when necessary to handle the transitions between regular regions, which are the ones that are topologically and geometrically more challenging.

A spline basis on a regular lattice has major advantages over traditional FEM elements, since it has the potential to be *both* accurate and efficient: It has a single degree of freedom (dof) per element, except at the boundary, yet it has full approximation power corresponding to the degree of the spline. This observation is one of the foundations of *isogeometric analysis* in three dimensions (3D) (Cottrell et al. 2009; Hughes et al. 2005). Unfortunately, it is easy to define and implement only for fully regular grids, which is not practical for most input geometries. The next best thing is spline bases on *pure hexahedral* meshes: While smooth constructions for polar configurations exist (Toshniwal et al. 2017), a solution applicable to general hexahedral meshes whose interior singular curves meet is still elusive, restricting this construction to simple shapes. Padded hexahedral-meshes (Maréchal 2009) are necessary to ensure a good boundary approximation for both regular and polycube (Tarini et al. 2004) hexahedral meshing methods, but they unfortunately cannot be used by these constructions, since their interior curve singularities meet in the padding layer.

We propose a hybrid construction that sidesteps these limitations: We use spline elements only on fully regular regions and fill the elements that are touching singular edges, or that are not hexahedra, with local constructions (harmonic elements for polyhedra, triquadratic polynomial elements for hexahedra). This construction further relaxes requirements for meshing, since it works on general hexahedral meshes (without any restriction on their singularity structure) but also directly supports *hex-dominant* meshes, which can be robustly generated with modern field-aligned methods (Gao et al. 2017a; Sokolov et al. 2016). These meshes consist mostly of well-shaped hexahedra with locally regular mesh structure but also contain other general polyhedra. Our construction takes advantage of this high regularity, adding a negligible overhead over the spline FEM basis only for the sparse set of non-regular elements.

We demonstrate that our proposed *Poly-Spline* FEM retains, to a large extent, both the approximation and performance benefits of splines, at the cost of the increasing basis construction complexity, and, at the same time, works for a class of meshes that can be robustly generated for most shapes with existing meshing algorithms.

Our method exhibits cubic convergence on a large dataset (see Figure 1(ref) for a selection), for a degree of freedom budget comparable to trilinear hexahedral elements, which have only quadratic convergence. To the best of our knowledge, this article is the first FEM method exploiting the advantages of spline basis that has been validated on a large collection of complex geometries.

2 RELATED WORK

When numerically solving PDEs using the finite-element method, one has to discretize the spatial domain into finite elements and define shape functions on these elements. Since shape functions, element types, and mesh generation are closely related, we discuss the relevant approaches in tandem.

For complex spatial domains, the discretization is frequently based on the Delaunay triangulation (Shewchuk 1996) or Delaunay tetrahedrization (Si 2015), respectively, since those tessellations can be computed in a robust and automatic manner. Due to their simplicity and efficiency, linear shape functions over triangular or tetrahedral elements are often the default choice for graphics applications (Hughes 2000), although they are known to suffer from locking for stiff PDEs, such as nearly incompressible elastic materials (Hughes 2000).

This locking problem can be avoided by using bilinear quadrangular or trilinear hexahedral elements (Q_1 elements), which have the additional advantage of yielding a higher accuracy for a given number of elements (Benzley et al. 1995; Cifuentes and Kalbag 1992). Triquadratic hexahedral elements (Q_2) provide even higher accuracy and faster convergence under mesh refinement (cubic converge in L_2-norm for Q_2 vs. quadratic converge for Q_1), but their larger number of degrees of freedom (8 vs. 27) leads to high memory consumption and computational cost.

The main idea of isogeometric analysis (IGA) (Cottrell et al. 2009; Engvall and Evans 2017; Hughes et al. 2005) is to employ the same spline basis for defining the CAD geometry as well as for performing numerical analysis. Using quadratic splines on

hexahedral elements results in the same cubic convergence order as Q_2 elements but at the much lower cost of *one* degree of freedom per element (comparable to Q_1 elements). This efficiency, however, comes at the price of a very complex implementation for non-regular hexahedral meshes. Moreover, generating IGA-compatible meshes from a given general boundary surface is still an open problem (Aigner et al. 2009; Li et al. 2013; Martin and Cohen 2010).

Concurrent work (Wei et al. 2018) introduces a construction that can handle irregular pure hex meshes, with tensor-product cubic splines used on regular parts. However, we focus on handling general polygonal meshes, and we use quadratic splines (note that our approach can be easily extended to cubic polynomials if desired).

A standard method for volumetric mesh generation is through hierarchical subdivision of an initial regular hexahedral mesh, leading to so-called octree meshes (Ito et al. 2009; Maréchal 2009; Zhang et al. 2013). The T-junctions resulting from adaptive subdivision can be handled by using T-splines (da Veiga et al. 2011; Sederberg et al. 2004) as shape functions. While this meshing approach is very robust, it has problems representing geometric features that are not aligned with the principal axes.

Even when giving up splines or T-splines for standard Q_1/Q_2 elements, the automatic generation of the required hexahedral meshes is problematic. Despite the progress made in this field over the past decade, *automatically* generating pure hexahedral meshes that (i) have sufficient element quality, (ii) are not too dense, and (iii) align to geometric features is still unsolved. Early methods based on paving or sweeping (Owen and Saigal 2000; Shepherd and Johnson 2008; Staten et al. 2005; Yamakawa and Shimada 2003) require complicated handling of special cases and generate too many singularities. Polycube methods (Fang et al. 2016; Fu et al. 2016; Gregson et al. 2011; Huang et al. 2014; Li et al. 2013; Livesu et al. 2013), field-aligned methods (Huang et al. 2011; Jiang et al. 2014; Li et al. 2012; Nieser et al. 2011), and the surface foliation method (Lei et al. 2017) are interesting research venues, but they are currently not robust enough and often fail to produce a valid mesh.

However, if the strict requirement of producing hexahedral elements only is relaxed, then field-aligned methods (Gao et al. 2017a; Sokolov et al. 2016) can robustly and automatically create hex-dominant polyhedral meshes, that is, meshes consisting of mostly, but not exclusively, of hexahedral elements. The idea is to build local volumetric parameterizations aligned with a specified directional field and construct the mesh from traced isolines of that parameterization, inserting general polyhedra if necessary. Their drawback is that the resulting *hex-dominant* meshes are not directly supported by most FEM codes.

One option is to split these general polyhedra into standard elements, leading to a mixed FEM formulation. For instance, the field-aligned meshing of Sokolov et al. (2016) extract meshes that are composed of hexahedra, tetrahedra, prisms, and pyramids. However, the quality of those split elements is hard to control in general. An interesting alternative is to avoid the splitting of polyhedra and instead incorporate them into the simulation, for instance, through mimetic finite differences (Lipnikov et al. 2014), the virtual element method (Beirão Da Veiga et al. 2013), or polyhedral finite elements (Manzini et al. 2014). The latter employ generalized barycentric coordinates as shape functions, such as mean value coordinates (Floater et al. 2005; Ju et al. 2005), harmonic coordinates (Joshi et al. 2007), or minimum entropy coordinates (Hormann and Sukumar 2008). From those options, harmonic coordinates seem most suitable, since they generalize both linear tetrahedra and trilinear hexahedra to general non-convex polyhedra (Bishop 2014; Martin et al. 2008). While avoiding splitting or remeshing hex-dominant meshes, the major drawback of polyhedral elements is the high cost for computing and integrating their shape functions.

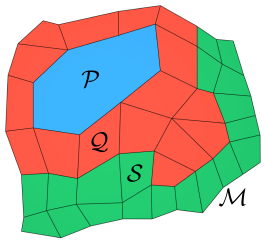


Fig. 2. Complexes involved in our construction. In green we show $\mathcal{S}$, in red $\mathcal{Q}$, and in blue $\mathcal{P}$.

In the above methods, the meshing stage severely restricts admissible shape functions, or the element type puts (too) strong requirements on the meshing. In contrast, we use the most efficient elements where possible and the most flexible elements where required, which enables the use of robust and automatic hex-dominant mesh generation.

3 ALGORITHM OVERVIEW

In this section, we introduce the main definitions we use in our algorithm description and outline the structure of the algorithm. We refer to Appendix A for a brief introduction to the finite-element method and the setup of our mathematical notation.

Input Complex and Subcomplexes. The input to our algorithm is a 3D polyhedral complex $\mathcal{M}$, with vertices $\mathbf{v}_i \in \mathbb{R}^3$, $i = 1, \ldots, N_V$, consisting of polyhedral cells C_i, $i = 1, \ldots, N_C$, most of which are hexahedra. Figure 2 shows a two-dimensional example of such complex. The edges, faces, and cells of the mesh are defined combinatorially, that is, edges are defined by pairs of vertices, faces by sequences of edges, and cells by closed surface meshes formed by faces. We assume that 3D positions of vertices are also provided as input and that $\mathcal{M}$ is three-manifold, i.e., that there is a way to identify vertices, edges, faces, and cells with points, curves, surface patches and simple volumes, such that their union is a three-manifold subset of $\mathbb{R}^3$.

We assume that for any hexahedron there is at most one non-hexahedral cell sharing one of its faces or edges, which can be achieved by refinement. We also assume that no two polyhedral cells are adjacent and that no polyhedron touches the boundary, which can also be achieved by merging polyhedral cells and/or refinement. This preprocessing step (i.e., one step of uniform refinement) is discussed in Section 6. As a consequence of our refinement, *all faces of $\mathcal{M}$ are quadrilateral.*

One of the difficulties of using general polyhedral meshes for basis constructions is that, unlike the case of, for example, pure tetrahedral meshes, there is no natural way to realize all elements of the mesh in 3D just from vertex positions (e.g., for a tetrahedral mesh, linear interpolation for faces and cells is natural). This

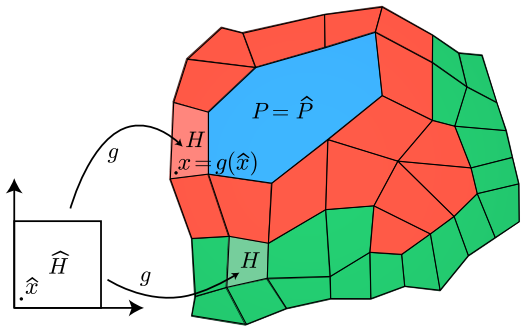


Fig. 3. Illustration of the geometric mapping.

requires constructing bases on an explicitly defined parametric domain associated with the input complex.

For this purpose, we define a certain number of complexes related to the original complex $\mathcal{M}$ (Figure 2). There are two goals for introducing these: defining the parametric domain for the basis and defining the *geometric map*, which specifies how the complex is realized in three-dimensional *physical space*.

- $\mathcal{H} \subseteq \mathcal{M}$ is the hexahedral part of $\mathcal{M}$, consisting of hexahedra H.
- $\mathcal{P} = \mathcal{M} \setminus \mathcal{H}$ is the non-hexahedral part of $\mathcal{M}$, consisting of polyhedra P.
- $\mathcal{S} \subseteq \mathcal{H}$ is the complex consisting of spline-compatible hexahedra S defined in Section 4.1.
- $\mathcal{Q} = \mathcal{H} \setminus \mathcal{S}$ is the spline-incompatible pure-hexahedral part of $\mathcal{M}$.

Note that the sub-complexes of $\mathcal{M}$ are nested: $\mathcal{S} \subseteq \mathcal{H} \subseteq \mathcal{M}$.

In the context of finite elements, the distinction between *parametric space* and *physical space* is critical: The bases on the hexahedral part of the mesh are defined in terms of parametric space coordinates, where all hexahedra are unit cubes; this makes it possible to define simple, accurate, and efficient bases. However, the derivatives in the PDE are taken with respect to physical space variables, and the unknown functions are naturally defined on the physical space. Remapping these functions to the parametric space is necessary to discretize the PDE using our basis. We define parametric domains $\widehat{\mathcal{M}}$, $\widehat{\mathcal{H}}$, $\widehat{\mathcal{S}}$, and $\widehat{\mathcal{Q}}$ corresponding to $\mathcal{M}$, $\mathcal{H}$, $\mathcal{S}$, and $\mathcal{Q}$, respectively. $\widehat{\mathcal{H}}$ consists of unit cubes $\widehat{H}$, one per hexahedron H with corresponding faces identified, and $\widehat{\mathcal{S}}$ and $\widehat{\mathcal{Q}}$ are its subcomplexes. The complete parametric space $\widehat{\mathcal{M}}$ is obtained by adding a set of polyhedra for $\mathcal{P}$, defined using the geometric map as described below. For polyhedra, physical and parametric space coincide.

Geometric Map and Complex Embedding. The input complex, as it is typical for mesh representations, does not define a complete geometric realization of the complex: Rather, it only includes vertex positions and element connectivity. We define a complete geometric realization as the *geometric map* $\mathbf{g} : \widehat{\mathcal{M}} \to \mathbb{R}^3$, from the parametric domain $\widehat{\mathcal{M}}$ to the physical space. We use $\hat{\mathbf{x}}$ for points in the parametric domain, and $\mathbf{x}$ for points in the physical space, and denote the image of the geometric map by $\Omega = \mathbf{g}(\widehat{\mathcal{M}})$ (Figure 3).

The definition requires bootstrapping: $\mathbf{g}$ is first defined on $\widehat{\mathcal{H}}$. For example, the simplest geometric map $\mathbf{g}$ on $\widehat{\mathcal{M}}$ can be obtained by trilinear interpolation: $\mathbf{g}$ restricted to a unit cube $\widehat{H} \subset \widehat{\mathcal{M}}$ is a trilinear interpolation of the positions of the vertices of its associated hexahedron H. We make the following assumption about $\mathbf{g}(\widehat{\mathcal{H}})$: The map is bijective on the faces of $\mathcal{H}$, corresponding to the boundary of any polyhedral cell P, and the union of the images of these faces does not self-intersect and encloses a volume P'. Section 6 explains how this is ensured. Then we complete $\widehat{\mathcal{M}}$ by adding the volume P' as the parametric domain for P. We add this volume to the parametric domain $\widehat{\mathcal{M}}$, identifying corresponding faces with faces in $\widehat{\mathcal{H}}$ and defining the geometric map to be the identity on these domains.

The simplest trilinear map is adequate for elements of the mesh outside the regular part $\mathcal{S}$ but is insufficient for accuracy on the regular part, as discussed below. We consider a more complex definition of $\mathbf{g}$ ensuring C^1 smoothness across interior edges and faces of $\mathcal{S}$, described in Section 5, after we describe our basis construction. Our construction is *isoparametric*, that is, it uses the same basis for the geometric map as for the solution.

In other words, on $\mathcal{Q}$ we use the standard tri-quadratic geometric map that maps each reference cube $[0,1]^3$ to the actual hex-element in the mesh. On $\mathcal{S}$ we use a C^1 spline mapping, explained in Section 5. On the polyhedral part, the geometric map is the identity, thus all quantities are defined directly on the physical domain.

Overview of the Basis and Discretization Construction. Given an input complex $\mathcal{M}$, we construct a set of bases $\widehat{\phi}_i : \widehat{\mathcal{M}} \to \mathbb{R}$, $i = 1, \ldots, N$, such that:

- the restriction of basis function $\widehat{\phi}_i$ to spline compatible hexahedral domains $\widehat{S} \in \widehat{\mathcal{S}}$ is a spline basis function;
- the restriction to hexahedra $\widehat{Q} \in \widehat{\mathcal{Q}}$ is a standard triquadratic (Q_2) element function;
- the restriction to polyhedra $\widehat{P} \in \widehat{\mathcal{P}}$ (or $P \in \mathcal{P}$) is a harmonic-based nonconformal, third-order accurate basis function.

The dofs corresponding to basis functions $\widehat{\phi}_i$ are associated with:

- each hexahedron either in $\mathcal{S}$ or adjacent to a spline-compatible one (*spline cell dofs*);
- each boundary vertex, edge, or face of $\mathcal{S}$ (*spline boundary dofs*); these are needed to have correct approximation on the boundary;
- each vertex, edge, face, and cell of $\mathcal{Q}$ (*triquadratic element degrees of freedom*).

The total number of degrees of freedom is denoted by N. While most of the construction is independent of the choice of PDE (we assume it to be second order), with a notable exception of the consistency condition for polyhedral elements, we use the Poisson equation to be more specific.

Note that hexahedra adjacent to $\mathcal{S}$, but not in $\mathcal{S}$ (i.e., hexahedra in $\mathcal{Q}$), get both spline dofs and triquadratic element dofs: Such a cell may have ≥28 dofs instead of 27.

Polyhedral cells are not assigned separate degrees of freedom: The basis functions with support overlapping polyhedra are those associated with dofs at incident hexahedra.

We assemble the standard stiffness matrix for an elliptic PDE, element-by-element, performing integration on the hexahedra $\widehat{H}$ of $\widehat{\mathcal{M}}$ and polyhedra P. The entry K_{ij} of the stiffness matrix $\mathbf{K}$ for the Poisson equation is computed as follows:

$$K_{ij} = \sum_{\widehat{C} \in \widehat{\mathcal{M}}} \int_{\mathbf{g}(\widehat{C})} \nabla \phi_i(\mathbf{x}) \cdot \nabla \phi_j(\mathbf{x}) \, \mathrm{d}\mathbf{x}, \tag{1}$$

where $\phi_i = \widehat{\phi}_i \circ \mathbf{g}^{-1}$. The actual integration is performed on the elements in the parametric domain $\widehat{\mathcal{M}}$, using a change of variables $\mathbf{x} = \mathbf{g}(\widehat{\mathbf{x}})$ for every element:

$$K_{ij} = \sum_{\widehat{C} \in \widehat{\mathcal{M}}} \int_{\widehat{C}} \nabla \widehat{\phi}_i(\widehat{\mathbf{x}})^{\mathsf{T}} \, \mathbf{A}(\widehat{\mathbf{x}}) \, \nabla \widehat{\phi}_j(\widehat{\mathbf{x}}) \, |\mathrm{D}\mathbf{g}| \, \mathrm{d}\widehat{\mathbf{x}}, \tag{2}$$

where $\mathbf{A}(\widehat{\mathbf{x}})$ is the metric tensor of the geometric map $\mathbf{g}$ at $\widehat{\mathbf{x}}$, given by $\mathrm{D}\mathbf{g}^{-1}\mathrm{D}\mathbf{g}^{-\mathsf{T}}$, with $\mathrm{D}\mathbf{g}$ being the Jacobian of $\mathbf{g}$.

In the next sections, we describe the construction of the basis on each element type, the geometric map, and the stiffness matrix construction.

4 BASIS CONSTRUCTION

We seek to construct a basis on $\Omega = \mathbf{g}(\widehat{\mathcal{M}})$ that has the following properties:

(1) It is C^0 everywhere on Ω, C^1 at regular edges and vertices, and C^∞ within each H and P (polynomials on hexahedra).
(2) It has approximation order 3 on each H and P.

The unknown function u on the domain Ω is approximated by $u_h = \sum_{i=1}^{N} u_i \phi_i$, where ϕ_i are the basis functions. The support of each basis function is a union of a set of the images under $\mathbf{g}$ of cells in $\widehat{\mathcal{M}}$.

The actual representation of the basis, which allows us to perform per-element construction of the stiffness matrix, consists of three parts. The first two parts are local: We define a *local* set of dofs and a *local* basis. For hexahedral elements, there are several types of local polynomial bases, each coming with its set of local dofs, associated with a local *control mesh* for the element. These basis functions are encoded as sets of polynomial coefficients. For polyhedral elements, all local basis functions are weighted combinations of harmonic kernel functions and a triquadratic polynomial, so these are encoded as kernel centers, weights, and polynomial coefficients.

The third part is the *local-to-global* linear map that represents local dofs in terms of the global ones. Importantly, unlike most standard FEM formulations, our local-to-global maps are not necessarily simply identifying local dofs the global ones: Some local dofs are linear combinations of global ones. These maps are formally represented by $m \times N$ matrices, where m is a small number of local dofs, and N is the total number of global dofs. However, as the elements local dofs depend only on nearby global dofs, these matrices have a small number of nonzeros and can be encoded in a compact form.

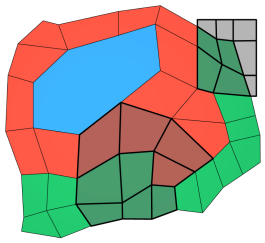


Fig. 4. Spline local grid (shown in dark) for an internal and a boundary quadrilateral. The color codes are as defined in Figure 2.

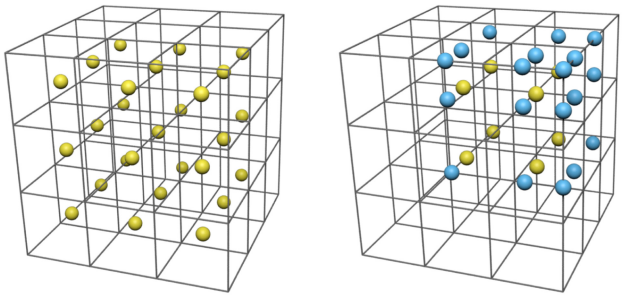


Fig. 5. Spline hex degrees of freedom for a central element and a corner one.

In the following, we consider the construction of these three elements (set of local basis functions, set of local dofs, local-to-global map) for each of our three element types. But before we can construct the basis for each element, hexahedral elements need to be classified into $\mathcal{S}$ (spline-compatible) and $\mathcal{Q}$.

4.1 Spline-compatible Hexahedral Elements

We define a hexahedron H to be spline compatible if its one-ring cell neighborhood is a $3 \times 3 \times 3$ regular grid, possibly cut on one or more sides if H is on the boundary, see Figure 4.

The local dofs of this element type form a $3 \times 3 \times 3$ grid (for interior elements), with the element in the center (Figure 5, left); for boundary elements, there are still 27 dofs, ensuring a full triquadratic polynomial reproduction. If a single layer with 9 dofs is missing, then we add an extra degree of freedom for each face of the local $3 \times 3 \times 2$ grid corresponding to the boundary. Other cases are handled in a similar manner; e.g., the configuration for a regular corner is shown in Figure 5, right.

The basis functions in this case are just the standard triquadratic uniform spline basis functions for interior hexahedra. For the boundary case, we use the knot vector $[0, 0, 0, 1, 2, 3]$ in the direction perpendicular to the boundary. Figure 6 shows an example of the bases in 2D, for an internal node on the left and for a boundary node on the right. Finally, the *local-to-global* map simply identifies local basis dofs with corresponding global ones.

Compared to a standard Q_2 element, the ratio of degrees of freedom to the number of elements is much lower (a single degree of freedom per element for splines), although the approximation order is the same.

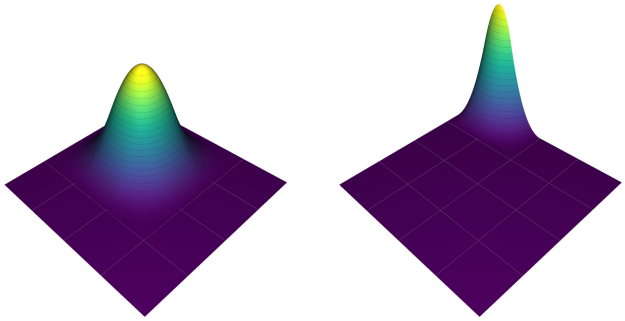


Fig. 6. Plot of the spline bases for a regular 2D grid.

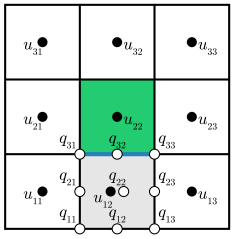


Fig. 7. Local-to-global map for a Q_2 element (gray) adjacent to a single spline element (green).

4.2 Q_2 Hexahedral Elements

This element is used for all remaining hexahedra. It is a standard element, widely used in finite-element codes. *Local dofs* for this element are associated with the element vertices, edge midpoints, face centers, and cell centers (Figure 26).

The *local basis functions* for the element are obtained as the tensor product of the interpolating quadratic bases on the interval $[0, 1]$, consisting of $(t - \frac{1}{2})(t - 1)$, $(t - \frac{1}{2})t$, and $(t - 1)t$ (Appendix C). The only complicated part in the case of Q_2 elements is the definition of the local-to-global map. For the two-dimensional setting, it is illustrated in Figure 7.

The difficulty in the construction of this map is due to the interface between spline elements and Q_2 elements and the need to ensure continuity between the two. In the two-dimensional case, suppose that a Q_2 element $Q \in \mathcal{Q}$ shares an edge with exactly one quad spline element $S \in \mathcal{S}$. Let u_{ij}, $i, j = 1, \ldots, 3$, be the global dofs of the spline element, and let q_{ij}, $i, j = 1, \ldots, 3$, be the degrees of freedom of the Q_2 element, as shown in the picture.

In this case, we ensure C^0 continuity of the basis by expressing the values of the polynomials on $Q \in \mathcal{Q}$ at the shared boundary points in terms of global degrees of freedom. Since both the Q_2 and the spline basis restricted to an edge are quadratic polynomials, they only need to be equal on three distinct points of the edge to ensure continuity. By noticing that the Q_2 basis is interpolatory at the nodes, it is enough to evaluate the spline basis at these edge nodes.

For the two-dimensional example in Figure 7, the local-to-global map for the local dofs q_{31}, q_{32}, and q_{33} along the edge (in blue) that

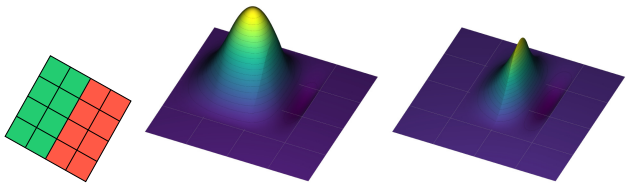


Fig. 8. Plot of the bases on a junction between a regular (green) and an irregular (red) part for a regular 2D grid.

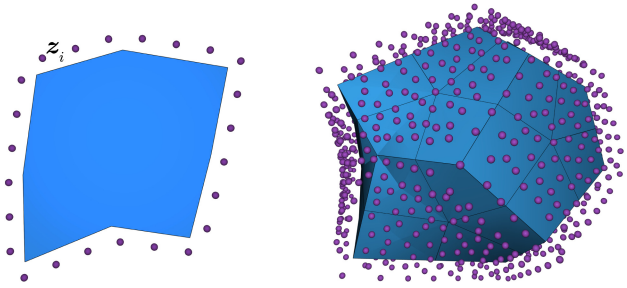


Fig. 9. The local basis for a polygon consists of the set of triquadratic polynomials q_d and harmonic kernels ψ_i centered at shown locations $\mathbf{z}_i$.

the Q_2 element shares with the spline is obtained as follows:

$$
\begin{aligned}
q_{31} &= \frac{1}{4}\left(u_{11} + u_{12} + u_{21} + u_{22}\right), \\
q_{32} &= \frac{3}{8}\left(u_{12} + u_{22}\right) + \frac{1}{16}\left(u_{11} + u_{21} + u_{13} + u_{23}\right), \\
q_{33} &= \frac{1}{4}\left(u_{12} + u_{13} + u_{22} + u_{23}\right).
\end{aligned}
\tag{3}
$$

In 3D, the construction is similar. We first identify all spline bases overlapping with a local dof q_{ij} on the boundary of a Q_2 element (i.e., a vertex, edge, or face dof). To determine the weights of the local-to-global map, we evaluate each spline basis on the local dof q_{ij} and set it as weight.

The remaining degrees of freedom of the Q_2 element are identified with global Q_2 degrees of freedom at the same locations. We note once again that at the center of cells in $\mathcal{Q}$ with neighboring cells in $\mathcal{S}$, there are *two* dofs, one spline dof and one Q_2 dof. Figure 8 shows an example of two basis functions on the transition from the regular part on the left to the "irregular" part on the right. We clearly see that on the regular part the bases are splines and on the irregular one are the standard Q_2 basis function: On the interface the functions are only C_0.

4.3 Basis Construction on Polyhedral Cells

The construction of the basis on the polyhedral cells is quite different from the construction of the basis on hexahedra. For hexahedra, the basis functions are defined on the parametric domain $\widehat{\mathcal{M}}$ and are remapped to $\Omega \subset \mathbb{R}^3$ via the geometric map. For polyhedra, we construct the basis directly in physical space.

On possible option to construct the basis on polyhedral cells is to split each polyhedral cells into tetrahedra. This approach has two main disadvantages: (i) It requires the use of pyramids to ensure conformity to the neighboring hexahedra, and (ii) it is difficult to guarantee a sufficient element quality after subdivision. Instead,

we follow the general approach of Martin et al. (2008) with two important alterations designed to ensure third-order convergence.

Recall that all polyhedron faces are quadrilateral, and all polyhedra are surrounded by hexahedra, specifically, Q_2 hexahedra as their neighborhood is not regular. Moreover, since we always perform an initial refinement step, there are no polyhedral cells touching each other. We use the degrees of freedom on the faces of these elements as degrees of freedom for the polyhedra; therefore, the *local-to-global* map in this case is trivial.

Each dof is already associated with a basis function ϕ_j defined on the hexahedra adjacent to the polyhedron. We construct the extension of ϕ_j to the polyhedron P from k harmonic kernels $\psi_i(\mathbf{x}) = \|\mathbf{x} - \mathbf{z}_i\|^{-1}$ centered at positions $\mathbf{z}_i$ outside the polyhedron and quadratic monomials $q_d(\mathbf{x})$, $d = 1, \ldots, 10$, as

$$\begin{aligned}\phi_j\big|_P(\mathbf{x}) &= \sum_{i=1}^{k} w_i^j \psi_i(\mathbf{x}) + \sum_{d=1}^{10} a_d^j q_d(\mathbf{x}) \\ &= \mathbf{w}^j \cdot \boldsymbol{\psi}(\mathbf{x}) + \mathbf{a}^j \cdot \mathbf{q}(\mathbf{x}),\end{aligned} \tag{4}$$

where $\mathbf{w}^j = (w_1^j, \ldots, w_k^j)^\mathsf{T}$, $\boldsymbol{\psi} = (\psi_1, \ldots, \psi_k)^\mathsf{T}$, $\mathbf{a}^j = (a_1^j, \ldots, a_{10}^j)^\mathsf{T}$, and $\mathbf{q} = (q_1, \ldots, q_{10})^\mathsf{T}$. The coefficients $\mathbf{w}^j$ and $\mathbf{a}^j$ are $r \times k$ and $r \times 10$ matrices, respectively, with $r = 1$ (scalar PDEs) or $r = 2, 3$ (vector PDEs). Following Martin et al. (2008), the weights $w_i^j, a_d^j \in \mathbb{R}^k$ are determined using a least-squares fit to match the values of the basis ϕ_j evaluated on a set of points sampled on the boundary of the polyhedron P.

In Martin (2011), it is shown that this construction automatically guarantees reproduction of linear polynomials if q_d are linear; the quadratic case is fully analogous. However, this condition is insufficient for high-order convergence, because our basis is *non-conforming*, that is, non C^0. In the context of the second-order PDEs we are considering, it means that it lacks C^0 continuity on the boundary of the polyhedron. For this type of element, additional *consistency conditions* are required to ensure high-order convergence. These conditions depend on the PDE that we need to solve.

FEM Theory Detour. To achieve higher-order convergence *three* conditions need to be satisfied: (1) polynomial reproduction; (2) consistency, which we discuss in more detail below; and (3) quadrature accuracy. We refer to standard FEM texts such as Braess (2007) for details, as well as to virtual element method literature (e.g., de Dios et al. (2016) is closely related).

To satisfy the third condition, we use high-order quadrature on the polyhedron: We decompose it into tetrahedra and use Gaussian quadrature points in each tetrahedron (the decomposition is detailed in Section 6.1). The first condition, polynomial reproduction, is ensured by construction of the basis above.

The second constraint, consistency, requires further elaboration. We first derive it for the Poisson equation and then summarize the general form. We leave as future work the complete proof of the convergence properties of our method (cf. de Dios et al. (2016)), which requires, in particular, a proper stability analysis. Nevertheless, in Section 7 we provide numerical evidence that our method does converge at the expected rate and that its conditioning is not affected in a significant way by the presence of nonconforming polyhedral elements.

The standard way to find the solution of a PDE for a finite-element system is to consider its weak form. For the Poisson equation, find u such that

$$\int_\Omega \Delta u\, v = -\int_\Omega \nabla u \cdot \nabla v = \int_\Omega f v, \qquad \forall v. \tag{5}$$

Remark. We omit, for readability, the integration variable $\mathrm{d}\mathbf{x}$. In the remaining formulas we use integration over the physical space exclusively, in practice carried over to the parametric space by adding the Jacobian of the geometric map.

Then, u is approximated by $u_h = \sum_i u_i \phi_i$, and v is taken to be in the space spanned by the basis functions ϕ_j. The stiffness matrix entries are obtained as $K_{ij} = \sum_C \int_C \nabla\phi_i \cdot \nabla\phi_j$, where the integral is computed per element C, leading to the discrete system $\mathbf{Ku} = \mathbf{f}$ (Appendix A).

For general non-conforming elements, however, we cannot rely on this standard approach. For example, if we consider piecewise-constant elements for the Poisson equation, then the stiffness matrix would be all zeros.

However, *for a given PDE*, one can construct converging non-conforming elements. One condition that is typically used is that the discrete matrix, constructed per element as above, gives us *exact* values of the weak-form integral for all polynomials reproduced by the basis (cf. k-consistency property in (de Dios et al. 2016)).

As our basis reproduces triquadratic monomials (i.e., they are in the span of bases ϕ_i), we have $q_d(\mathbf{x}) = \sum_i q_d^i \phi_i(\mathbf{x})$. To ensure consistency, we require that any *nonconforming basis function* ϕ_j satisfies

$$-\int_{\mathrm{g}(\widehat{\mathcal{M}})} \Delta q_d \phi_j = \sum_i K_{ij} q_d^i \tag{6}$$

for all triquadratic monomials q_d.

To convert this equation to an equation for the unknown coefficients w_i^j and a_d^j, we observe that

$$\sum_i K_{ij} q_d^i = \int_{\mathrm{g}(\widehat{\mathcal{M}})} \left(\sum_i q_d^i \nabla\phi_i\right) \cdot \nabla\phi_j = \int_{\mathrm{g}(\widehat{\mathcal{M}})} \nabla q_d \cdot \nabla\phi_j, \tag{7}$$

due to the polynomial reproduction property. Separating the integral into the part over the hexahedra $g(\widehat{\mathcal{M}} \setminus P)$ and over the polyhedron $P = g(P)$, we write

$$\begin{aligned}\sum_i K_{ij} q_d^i &= C_H + \int_P \nabla q_d \cdot \nabla\left(\mathbf{w}^j \cdot \boldsymbol{\psi} + \mathbf{a}^j \cdot \mathbf{q}\right) \\ &= C_H + \mathbf{b}^\mathsf{T}\mathbf{w}^j + \mathbf{c}^\mathsf{T}\mathbf{a}^j,\end{aligned} \tag{8}$$

where

$$C_H = \sum_{\widehat{C} \in \widehat{\mathcal{M}} \setminus P} \int_{\mathrm{g}(\widehat{C})} \nabla q_d \cdot \nabla\phi_j, \quad \mathbf{b} = \int_P \nabla q_d \cdot \nabla\boldsymbol{\psi},$$
$$\mathbf{c} = \int_P \nabla q_d \cdot \nabla\mathbf{q}.$$

Similarly, the left-hand side of Equation 6 is reduced to a linear combination of $\mathbf{w}^j$ and $\mathbf{a}^j$. This forms a set of additional constraints for the coefficients of the basis functions on the polyhedron. To enforce them on each polyhedron, we solve a constrained least-squares system for each nonconforming basis function and store the obtained coefficients.

Importantly, the addition of constraints to the least-squares system does not violate the polynomial reproduction property on the polyhedron. This can be seen as follows. Let v_h be the linear combination of basis functions ϕ_i overlapping P that yields a triquadratic mononomial q_d when restricted to P. Then v_h is continuous on Ω: The samples at the points of the boundary are from a quadratic function and, therefore, match exactly the quadratic continuation to adjacent hexahedra.

The consistency condition (Equation (6)) applied to v_h simply states that it satisfies the integration by parts formula, which it does as it is C^0 at the element boundaries and smooth on the elements:

$$-\sum_C \int_C \Delta q_d v_h = \sum_C \int_C \nabla q_d \cdot \nabla v_h.$$

We conclude that v_h is in the space defined by the consistency constraint, and imposing this constraint preserves polynomial reproduction. See Appendix D for the complete list of constraints for the Poisson equation.

More generally, for a linear PDE and for any polynomial q (for vector PDEs, e.g., elasticity, this means that all components are polynomial) we require

$$a(q_d, v_h) = a_h(q_d, v_h), \quad \text{where} \quad a(u,v) = \int_\Omega \mathcal{F}(x, u, \nabla u, \Delta u)v,$$

where $\mathcal{F}$ is a linear function of its arguments depending on u and a_h is defined as a sum of integral over Ω after formal integration by parts of $\mathcal{F}$ to eliminate the second-order derivatives. For a conforming C^0 basis, this condition automatically follows from the integration by parts formulas, which are applicable. We now split the two bilinear forms as $a = a^H + a^P$ and $a_h = a_h^H + a_h^P$, where a^H and a_h^H contains the integral over the hexahedral known part, and a^P and a_h^P the integral over the polyhedral unknown part. Thus, for a basis ϕ_j, we obtain the following set of constraints:

$$\begin{aligned} a^H(q_d, \phi_j) - a_h^H(q_d, \phi_j) = {} & a^H\left(q_d, \mathbf{w}^j \cdot \boldsymbol{\psi}(\mathbf{x}) + \mathbf{a}^j \cdot \mathbf{q}(\mathbf{x})\right) \\ & - a_h^H\left(q_d, \mathbf{w}^j \cdot \boldsymbol{\psi}(\mathbf{x}) + \mathbf{a}^j \cdot \mathbf{q}(\mathbf{x})\right). \end{aligned}$$

For a scalar-valued PDE, we have the same number of constraints (5 in 2D and 9 in 3D) as monomials q_d, thus we are guaranteed to have a solution that respects the constraints for any $k > 0$. For vector PDEs (e.g., elasticity), we impose the additional constraints such that the coefficients $(w_i^j)_\alpha$ are the same for all dimensions $\alpha = 1, \ldots, r$, $r = 2$ or $r = 3$, which simplifies the implementation but increases the number of required centers $\mathbf{z}_j$, so that all constraints can be satisfied. More explicitly, for vector PDEs we require that the constraints

$$a(q_d^s \mathbf{e}_\alpha, \phi_j^s \mathbf{e}_\beta) = a_h(q_d^s \mathbf{e}_\alpha, \phi_j^s \mathbf{e}_\beta)$$

for $\alpha, \beta = 1, \ldots, r$ are satisfied, with q^s and ϕ_j^s denoting scalar polynomials and scalar basis functions, respectively, defined as in Equation (4) for dimension 1, and where $\mathbf{e}_\alpha$ is the unit vector for axis α. For dimensions 2 and 3, the number of monomials q is 5 and 9, respectively. The number of constraints is given by $r^2 q - q$, and thus we will need at least 15 $\mathbf{z}_i$ in 2D and 72 in 3D to ensure that the constraints are respected.

4.4 Imposing Boundary Conditions

We consider two standard types of boundary conditions: Dirichlet (fixed function values on the boundary) and Neumann (fixed normal derivatives at the boundary). Neumann (also known as natural) boundary conditions are handled in the context of the variational formulation of the problem as extra integral terms, in the case of inhomogeneous conditions. Homogeneous conditions do not require any special treatment and are imposed automatically in the weak formulation.

We assume that the Dirichlet conditions are given as a continuous function defined on the boundary of the domain. For all boundary dofs, we sample the boundary condition on the faces of the domain and perform a least-squares fit to retrieve the nodal values.

5 GEOMETRIC MAP CONSTRUCTION

The geometric map is a map from $\widehat{\mathcal{M}}$ to $\Omega \subset \mathbb{R}^3$, defined per element. Its primary purpose is to allow us to construct basis functions $\widehat{\phi}_i$ on reference domains (i.e., the elements of $\widehat{\mathcal{M}}$ that are unit cubes) and then to remap them to the physical space as $\phi_i = \widehat{\phi}_i \circ \mathbf{g}^{-1}$. As the local basis on the polyhedral elements is constructed directly in the physical space, $\mathbf{g}$ is the identity on these elements.

The requirements for the geometric map are distinct for the spline and Q_2 elements and are matched by using spline basis itself for $\mathcal{S}$ and trilinear interpolation for Q_2 elements.

Because of the geometric mapping $\mathbf{g}$, for the quadratic spline, the basis ϕ_i does not reproduce polynomials in the physical space; nevertheless, the approximation properties of the basis are retained (Bazilevs et al. 2006).

For Q_2 elements, Arnold et al. (2002) shows that *bilinear* maps are sufficient and in fact allow us to retain reproduction of triquadratic polynomials in the physical space. This is very important for the basis construction on polyhedral elements, as polynomial reproduction on these elements depends on reproduction of polynomials on the polyhedron boundary.

Computing the Geometry Map. If we assume that the input only has vertex positions $\mathbf{v}_i$ for $\mathcal{M}$, then we solve the equations $\mathbf{g}(\widehat{\mathbf{x}}_i) = \mathbf{v}_i$, which is a linear system of equations in terms of coefficients of $\mathbf{g}$ in the basis we choose. In the trilinear basis, the system is trivial, as the coefficients coincide with the values at $\mathbf{x}_i$, and these are simply set to $\mathbf{v}_i$. For the triquadratic basis, this is not the case, and a linear system needs to be solved. If the system is under-determined, then we find the least-norm solution.

6 MESH PREPROCESSING AND REFINEMENT

Without loss of generality, we restrict the meshing discussion to 2D, as the algorithm introduced in this section extends naturally to 3D.

For the sake of simplicity, in this discussion the term *polygon* refers to non-quadrilateral elements. As previously mentioned, our method can be applied to hybrid meshes without two adjacent polygons and without polygons touching the boundary, which we ensure with one step of refinement. While our construction could be extended to support these configurations, we favored refinement due to its simplicity. Refining polygonal meshes is an interesting problem on its own: While there is a canonical way

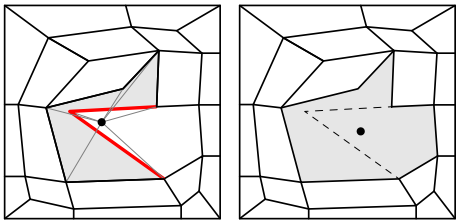


Fig. 10. Our algorithm iteratively merges polygons (gray polygon in the first image) until the barycenter of the merged polygon is inside its kernel (gray polygon in the second).

to refine quads, there are multiple ways to refine a polygon. We propose the use of polar refinement (Section 6.2), which has the added benefit of allowing us to resample large polygons to obtain a uniform element size. However, to avoid self-intersections between edges during the refinement, we impose each polygon be star shaped. This condition is often, but not always, satisfied by existing hybrid meshers: We thus introduce a simple merging and splitting procedure to convert hybrid meshes into star-shaped polyhedral meshes (Section 6.1) and then detail our refinement strategy (Section 6.2).

Another advantage of restricting ourselves to star-shaped polygons is that partitioning it into triangles (respectively, tetrahedra in 3D) is trivial by introducing a point in the kernel and connecting it to all the boundary faces. This step is required to generate quadrature points for the numerical integration (Section 4.3): The quality of the partitioning is usually low, but this is irrelevant for this purpose.

6.1 Mesh Preprocessing

We propose a simple and effective algorithm to convert polygonal meshes into star-shaped polygonal meshes by combining existing polygons until they are star shaped (and eventually splitting them if they contain a concave part of the boundary).

For every non-star-shaped polygon, we compute its barycenter and connect it to all its vertices (Figure 10, left). This procedure generates a set of intersecting segments (red in Figure 10), which we use to grow the polygon by merging it with the faces incident to each intersecting segment. The procedure is repeated until no more intersections are found, which usually happens in one or two iterations in our experiments. If we reach a concave boundary during the growing procedure, then it might be impossible to obtain a star-shaped polyhedron by merging alone: In these cases, we triangulate the polygon, and merge the resulting triangles in star-shaped polygons if possible.

6.2 Polar Refinement

Each star-shaped polygon is refined by finding a point in its kernel (Figure 11(a)), connecting it to all its vertices (Figure 11(b)), splitting each edge with mid-point subdivision and connecting them to the point in the kernel (Figure 11(c)), and, finally, adding rings of quadrilaterals around the boundary (Figure 11(d)). Figure 12 shows an example of polar refinement in two and three dimensions. The more splits are performed in the edge, the more elements are added. This is a useful feature to homogenize the element size

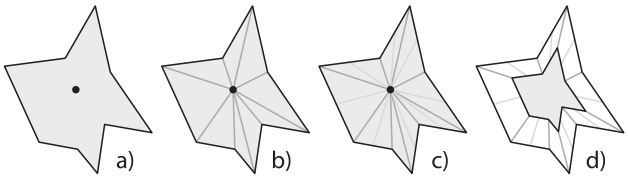


Fig. 11. Polar refinement for polygons.

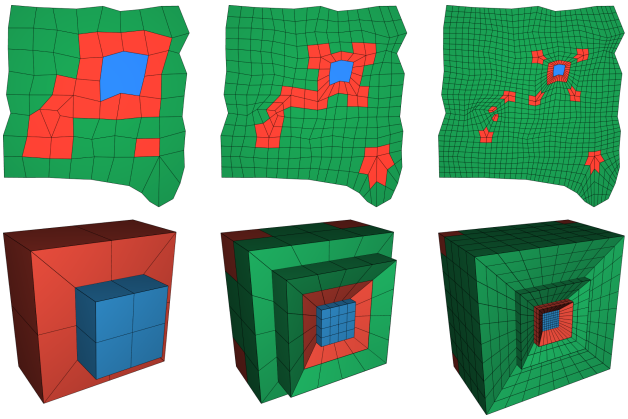


Fig. 12. Example of polar refinement for a polygon and a polyhedron. The bottom view is a cut-through of the actual 3D mesh.

in case the polygons were expanded too much during the mesh preprocessing stage. In our implementation, we split the edges evenly, ensuring that the shortest segment has a length as close as possible to the average edge length of the input mesh.

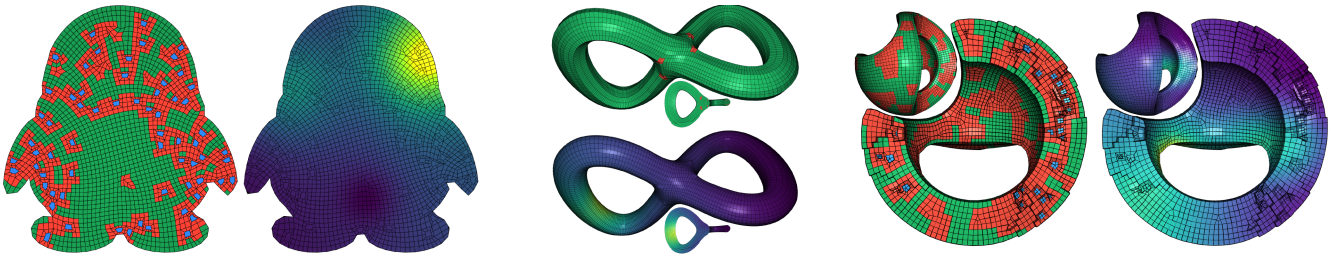
7 EVALUATION

We demonstrate the robustness of our method by solving the Poisson equation on a dataset of pure hex and hybrid meshes, consisting of 205 star-shaped polygonal meshes in 2D, 165 pure hexehedral meshes in 3D, and 29 star-shaped polyhedral meshes in 3D. The dataset can be found at https://cims.nyu.edu/gcl/papers/2019-Polyspline-Dataset.zip. All those meshes were automatically generated using Gao et al. (2017a, 2017b). We show a selection of meshes from our dataset in Figures 1 and 13.

We evaluated the performance, memory consumption, and running time of our proposed spline construction compared with standard Q_1 and Q_2 elements. For our experiments, we compute the approximation error on a standard Franke's test function (Franke 1979) in 2D and 3D (Appendix B). Note that in all these experiments, we enforced the consistency constraints on the bases spanning the polyhedral elements to ensure the proper convergence order.

The 2D experiments were run on a PC with an Intel Core i7-5930K CPU @ 3.50GHz with 64GB, while the 3D dataset was run on a HPC cluster with a memory limit of 64GB.

Absolute Errors. Figure 14 shows a scatter plot of the L_2 and L_∞ errors on both 2D and 3D datasets, with respect to the number of bases created by each type of elements (Q_1, Q_2, Splines), after one step of polar refinement. The plot shows that in 2D both the L_2 and L_∞ errors are about 1.5 orders of magnitude lower for our



2D polygon. 3D hexahedral mesh. 3D hybrid mesh.

Fig. 13. Solution of the Poisson problem different meshes.

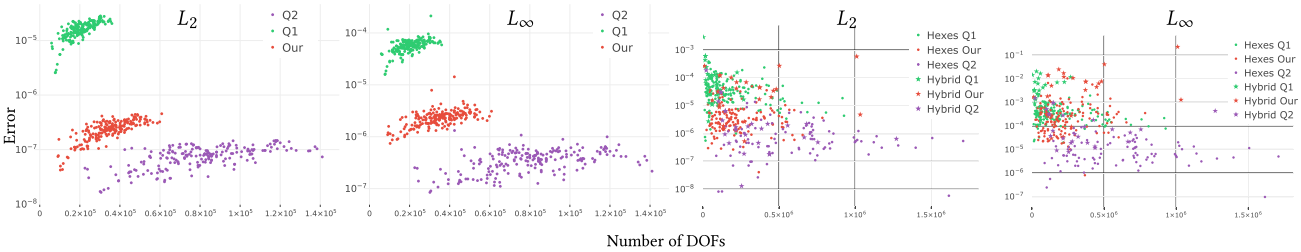


Fig. 14. Scatter plot of the L_2 and L_∞ error versus the number of dofs on the 2D (first two) and 3D (last two) dataset.

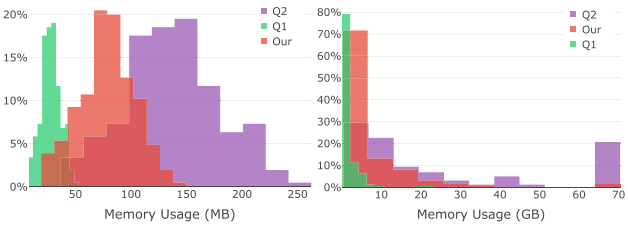


Fig. 15. Peak memory for the direct solver as reported by Pardiso. Left: Two-dimensional results. Right: Three-dimensional results.

splines compared to Q_1, while keeping a similar number of dofs. In comparison, Q_2 has lower error, but requires a much larger number of dofs. In 3D the spread of both errors is much larger, and the gain in L_∞ is less visible, but still present, compared to Q_1.

Memory. A histogram of the memory consumption of the solver is presented in Figure 15. The figure shows the peak memory usage as reported by Pardiso (Petra et al. 2014a, 2014b) when solving the linear system arising from the FEM. Of the 159 pure hexahedral models we tested, 33 went out of memory when solving using Q_2 elements, while only 2 are too big to solve with our spline bases. On the star-shaped hybrid meshes, one model is too big to solve for both Q_2 and our spline construction. More detailed statistics are reported in Table 1. We remark that the error for our method is higher than Q_2, because our method has fewer dofs (50% less in average), since both meshes have the same number of vertices.

Time. Figure 16 shows the assembly time and solve time for solving a Poisson problem on an unit square (cube) under refinement in two (three) dimensions. Note that both steps (assembly and solve) are performed in parallel. For the 2D experiment we used a 3.1GHz Intel Core i7-7700HQ with 8 threads, while in 3D we used a 3.5GHz Intel Core i7-5930K with 12 threads (both machines use hyper-threading). In Table 1 we summarize the timings for the large dataset using a 2.6GHz Intel Xeon E5-2690v4 with 8 threads. In all cases, the total time is dominated by the solving time.

Table 1. Dataset 3D Pure Hexahedra + Star-shaped Polyhedra (188 Models in Total)

		Num dofs	Solver	Bases	Assembly	Memory (MiB)	L_2 Error	L_∞ Error
Q_1	mean	174,335	0°8′26″	0°15′40″	0°0′37″	1,132	7.60e−05	1.11e−03
	std	177,192	0°20′29″	0°19′44″	0°0′40″	1,589	2.27e−04	2.69e−03
	min	3,035	0°0′0″	0°0′17″	0°0′1″	5	5.57e−07	1.98e−05
	median	105,451	0°1′2″	0°9′8″	0°0′23″	500	3.18e−05	3.39e−04
	max	926,938	0°182′32″	0°101′27″	0°5′20″	9,329	2.88e−03	2.09e−02
$Q_2^\star$	mean	552,583	0°63′43″	0°13′11″	0°0′55″	5,716	3.62e−06	6.34e−05
	std	355,783	0°72′42″	0°11′17″	0°0′59″	4,382	1.80e−05	2.00e−04
	min	21,525	0°0′5″	0°0′19″	0°0′3″	94	5.85e−09	9.58e−08
	median	457,358	0°34′17″	0°8′38″	0°0′40″	4,586	6.31e−07	1.06e−05
	max	1,709,712	0°289′19″	0°52′4″	0°6′56″	15,677	1.87e−04	1.50e−03
our*	mean	239,245	0°34′30″	0°14′59″	0°1′33″	3,728	1.65e−05	2.88e−03
	std	178,979	0°62′19″	0°12′34″	0°1′15″	3,787	5.57e−05	1.82e−02
	min	9,987	0°0′1″	0°0′21″	0°0′3″	61	4.09e−08	8.04e−07
	median	189,880	0°9′13″	0°10′13″	0°1′11″	2,391	3.46e−06	2.62e−04
	max	1,033,492	0°324′8″	0°55′11″	0°7′9″	15,681	5.85e−04	2.30e−01

The memory is the total peak memory (in MiB) as reported by the solver Pardiso. * *does not include the models that went out of memory.* From left to right, the total number of DOFs, the time required to solve the system, the time used to build the bases, the time employed to assembly the stiffness matrix, the peak memory, the L_2 error, and the L_∞ error.

Convergence. Figures 17 and 18 show the convergence of spline elements vs. Q_1 and Q_2 for the L_2, L_∞, and H_1 norms, in the ideal case of a uniform grid, in 2D and 3D. This is in a sense the best-case scenario that can be expected for our spline construction: Every

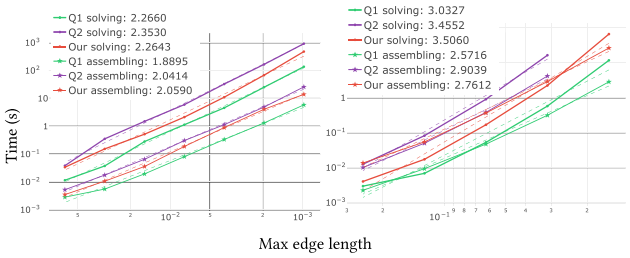


Fig. 16. Time required to assemble the stiffness matrix and solve the linear system on a regular grid in 2D (left) and 3D (right).

element is regular and has a 3^2 or 3^3 neighborhood. In this situation, splines exhibit a superior convergence >3.0 under both L_2, L_∞, and H_1 norms.

On a 2D test mesh with mixing polygons and splines (model shown in Figure 12 top), we achieved a convergence rate of 2.8 in L_∞ and 3.1 in L_2 (Figure 19 , left). Figure 19 also displays the convergence we obtained on a very simple hybrid 3D mesh, starting from a cube marked as a polyhedron, to which we applied our polar refinement described in Section 6. On this particular mesh, the splines exhibited a L_∞ convergence similar to Q_2, albeit producing an error that is somewhat larger.

Consistency Constraints. Figure 20 shows the effect of our consistency constraint on the convergence of a polygonal mesh under refinement (the one shown in Figure 12, top), with Q_2 elements used on the quadrilateral part. Without imposing any constraint on the bases overlapping the polygon, one can hope at best for a convergence of ~2.0, whereas pure Q_2 elements should have a convergence rate of 3.0. With a constraint ensuring linear reproduction for the bases defined on polyhedra, the convergence rate is still only ~2.5. Finally, with the constraints we describe in Section 4.3 to ensure the bases reproduce triquadratic polynomials, we reach the expected convergence rate of ~3.0.

Polyhedral Basis Resilience. Our polyhedral bases are less susceptible to badly shaped elements than Q_2. We computed the L_2 and L_∞ interpolation errors for the gradients of the Franke function for 14 badly shaped hexahedra, Figure 21 shows some of them. The L_2 and L_∞ maximum and average errors are 3 times smaller with our polygonal basis.

Conditioning and Stability. An important aspect of our new FE method is the conditioning of the resulting stiffness matrix: This quantity relates to both the stability of the method and to its performances when an iterative linear solver is used (important only for large problems where direct solvers cannot be used due to their memory requirements). We compute the condition number of the Poisson stiffness matrix on a regular and perturbed grid (Figure 22). In both cases, our discretization has a good conditioning number, slightly higher than pure linear elements, but lower than pure quadratic elements (while sharing the same cubic convergence property). To evaluate the conditioning of the polyhedral bases, we started from a base mesh of good quality, marked 5% of the quads as polygons, and pushed one of the vertices inward. Even for this extreme distortion of polyhedral elements, the conditioning remained similar to the case when no polyhedral elements are used on the same mesh.

Elasticity. While most of our testing was done for the Poisson equation, we have performed some testing of linear elasticity problems. Figure 23 top shows the solution of a linear elasticity problem on a pure hexahedral mesh. The outer loops of the knots are pulled outside of the figure, deforming the knot. The color in the figure represents the magnitude of the displacement vectors. On the bottom, we show the result for a Young's modulus of 2e5.

Figure 24 shows a plot for the linear elasticity PDE with Young's modulus 200 and Poisson's ratio 0.35 on a regular grid, and similar results are obtained an hybrid mesh, Figure 25. The convergence plots for Q_1 and Q_2 are obtained by mixing regular Q_1/Q_2 bases with the polyhedral construction (Section 4.3).

8 LIMITATIONS AND CONCLUDING REMARKS

We introduced Poly-Spline FEM, an integrated meshing and finite-element method designed to take advantage of recent developments in hexahedral-dominant meshing, opening the doors to black-box analysis with a high-order basis and cubic convergence under refinement. Our approach is to use the best possible basis for each element of the mesh and is amenable to continuous improvement, as the mesh generation methods and basis constructions improve. For instance, in this setting, one can avoid costly manual mesh repair and improvement, at the expense of modest increases in solution time, by switching to more expensive, but much less shape-sensitive, elements when a hexahedron is badly shaped.

While our basis construction is resilient to bad element quality, the geometric map between the splines and the Q_2 elements might introduce distortion (and even inversions in pathological cases), lowering convergence rate. These effects could be ameliorated by optimizing the positions of the control points of the geometric map, which is an interesting avenue for future work.

Our current construction always requires an initial refinement step to avoid having polyhedra adjacent to other polyhedra or to the boundary. This limitation could be lifted by generalizing our basis construction and would allow our method to process very large datasets that cannot be refined due to memory considerations. Another limitation of our method is that the consistency constraints in our basis construction (Section 4.3) are PDE dependent, and they thus require additional effort to be used with a user-provided PDE: a small and reusable investment compared to the cost of manually meshing with hexahedra every surface that one wishes to analyze using Q_2 elements. The code can be found at https://polyfem.github.io/ and provides an automatic way to generate such constraints relying on both the local assembler and automatic differentiation.

Poly-Spline FEM is a practical construction in-between unstructured Q_2 and fully-structured pure splines: It requires a smaller number of dofs than Q_2 (thanks to the spline elements) while preserving cubic convergence rate. We believe that our construction will stimulate additional research in the development of heterogeneous FEM methods that exploit the regularity of spline basis and combine it with the flexibility offered by traditional FEM elements. To allow other researchers and practitioners to immediately build

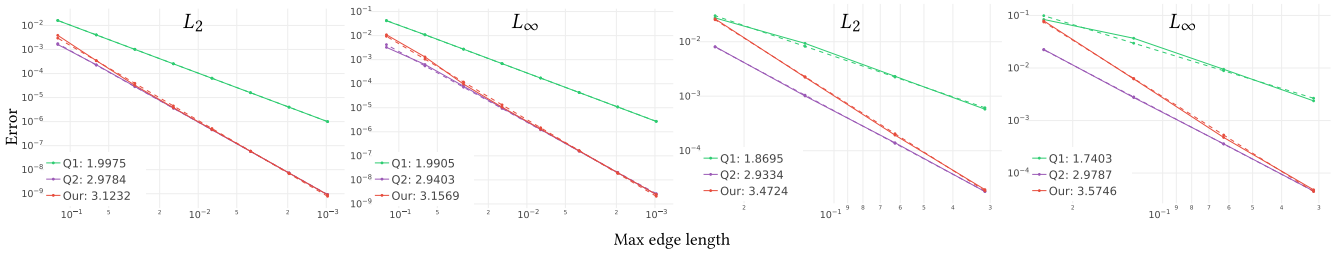


Fig. 17. Poisosn equation convergence plot in L_2 and L_∞ norm on a regular grid in 2D (first two) and 3D (last two).

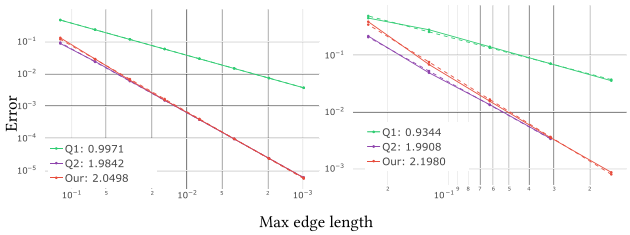


Fig. 18. Poisson equation convergence plot in H_1 norm on a regular grid in 2D (first plot) and 3D (second plot).

upon our construction, we will release our entire software framework as an open-source project.

APPENDICES

A BRIEF FINITE ELEMENT INTRODUCTION

Many common elliptic partial differential equations have the general form

$$\mathcal{F}(\mathbf{x}, u, \nabla u, \Delta u) = f(\mathbf{x}), \qquad \mathbf{x} \in \Omega,$$

subject to

$$u(\mathbf{x}) = d(\mathbf{x}), \mathbf{x} \in \partial\Omega_D \qquad \text{and} \qquad \nabla u(\mathbf{x}) \cdot \mathbf{N}(\mathbf{x}) = n(\mathbf{x}), \mathbf{x} \in \partial\Omega_N,$$

where $\mathbf{N}(\mathbf{x})$ is the surface normal, $\partial\Omega_D$ is the Dirichlet boundary where the function u is constrained (e.g., positional constraints), and $\partial\Omega_N$ is the Neumann boundary where the gradient of the function u is constrained. The most common PDE in this class is the Poisson equation $-\Delta u = f$.

Weak Form. The first step in a finite-element analysis consists of introducing the weak form of the PDE: Find u such that

$$\int_\Omega \mathcal{F}(\mathbf{x}, u, \nabla u, \Delta u)\, v(\mathbf{x})\, \mathrm{d}\mathbf{x} = \int_\Omega f(\mathbf{x})\, v(\mathbf{x})\, \mathrm{d}\mathbf{x}$$

holds for any *test function* v vanishing on the boundary. This reformulation has two advantages: (1) It simplifies the problem, and (2) it weakens the requirement on the function u. For instance, in case of the Poisson equation, the strong form is well defined only if u is twice differentiable, which is a difficult condition to enforce on a discrete tesselation. However, the weak form requires only that the second derivatives of u are integrable, allowing discontinuous jumps. Using integration by parts, it can be further relaxed to

$$\int_\Omega \nabla u(\mathbf{x}) \cdot \nabla v(\mathbf{x})\, \mathrm{d}\mathbf{x} = \int_\Omega f(\mathbf{x})\, v(\mathbf{x})\, \mathrm{d}\mathbf{x},$$

where only the gradient of u needs to be integrable, that is, $u \in H^1$, and can thus be represented using piecewise-linear basis functions.

Basis Functions. The key idea of a finite-element discretization is to approximate the solution space via a *finite* number of basis functions ϕ_i, $i = 1, \ldots, N$, which are independent from the PDE we are interested in. The number of nodes (and basis functions) per element and their position is directly correlated to the order of the basis, see Figure 26. We note that the nodes coincide the mesh vertices only for linear basis functions. Instead of solving the PDE, the goal becomes finding the coefficients u_i, $i = 1, \ldots, N$ of the discrete function $u_h(\mathbf{x}) = \sum_{i=1}^N u_i \phi_i(\mathbf{x})$ that approximates the unknown function u. For a linear PDE, this results in a linear system $\mathbf{Ku} = \mathbf{f}$, where $\mathbf{K}$ is the $N \times N$ stiffness matrix, $\mathbf{f}$ captures the boundary conditions, and $\mathbf{u}$ is the vector of unknown coefficients u_i. For instance, for the Laplace equation the entries of the stiffness matrix are

$$K_{ij} = \int_\Omega \nabla\phi_i(\mathbf{x}) \cdot \nabla\phi_j(\mathbf{x})\, \mathrm{d}\mathbf{x}.$$

Local Support. Commonly used basis functions are locally supported. As a result, most of the pairwise intgerals are zero, leading to a sparse stiffness matrix. The pairwise integrals can be written as a sum of integrals over the elements (e.g., quads or hexes) on which both functions do not vanish. This representation enables so-called *per-element assembly*: For a given element, a local stiffness matrix is assembled.

For instance, if an element C has four non-zero basis functions ϕ_i, ϕ_j, ϕ_k, ϕ_l (this is the case for linear Q_1 quad), then the local stiffness matrix $\mathbf{K}^L \in \mathbb{R}^{4\times4}$ for the Poisson equation is

$$K^L_{o,p} = \int_C \nabla\phi_n(\mathbf{x}) \cdot \nabla\phi_m(\mathbf{x})\, \mathrm{d}\mathbf{x},$$

where $o, p = 1, \ldots, 4$ and $m, n \in \{i, j, k, l\}$. By using the mapping of local indices (o, p) to global indices (m, n), the local stiffness matrix entries are summed to yield the global stiffness matrix entries.

Geometric Mapping. The final piece of a finite-element discretization is the geometric mapping $\mathbf{g}$. The local integrals need to be computed on every element. The element stiffness matrix entries are computed as integrals over a *reference* element $\hat{C}$ (e.g., a regular unit square/cube) through change of variables,

$$\int_C \nabla\phi_n(\mathbf{x}) \cdot \nabla\phi_m(\mathbf{x})\, \mathrm{d}\mathbf{x} = \int_{\hat{C}} (\mathrm{Dg}^{-\mathsf{T}} \nabla\hat{\phi}_n(\mathbf{x})) \cdot (\mathrm{Dg}^{-\mathsf{T}} \nabla\hat{\phi}_m(\mathbf{x})) |\mathrm{Dg}|\, \mathrm{d}\mathbf{x},$$

where Dg is the Jacobian matrix of the geometric mapping $\mathbf{g}$ and $\hat{\phi} = \phi \circ \mathbf{g}$ are the bases defined on the reference element $\hat{C}$. While usually $\mathbf{g}$ is expressed by linear combination of ϕ_i, leading to

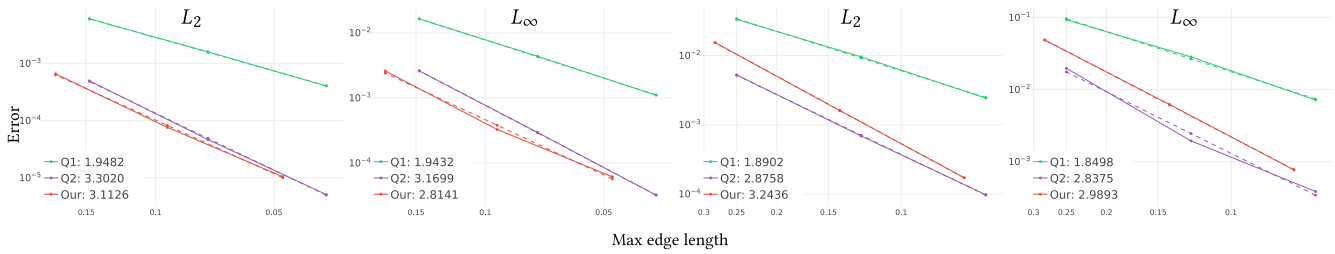


Fig. 19. Poisson equation convergence plot in L_2 and L_∞ norm for a hybrid mesh in 2D (first two) and 3D (last two). Meshes are show in Figure 12.

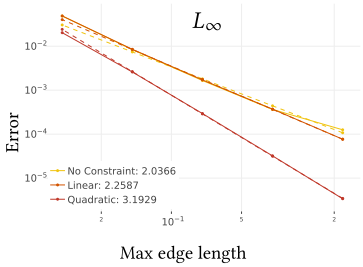


Fig. 20. L^∞ convergence for the different consistency constraints on the polyhedron of Figure 12.

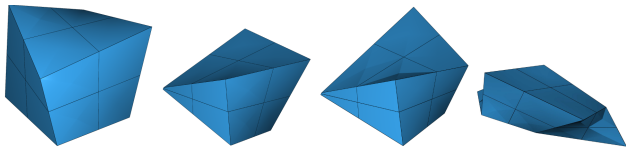


Fig. 21. Low-quality polyhedra used to evaluate the interpolation errors.

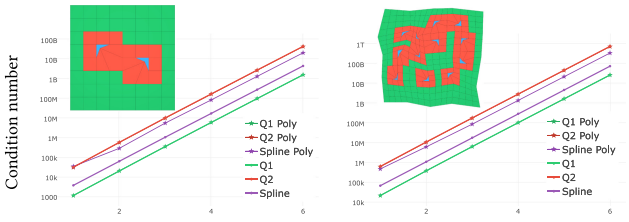


Fig. 22. Evolution of the condition number of the stiffness matrix for the Poisson problem under refinement. For each level of refinement, we artificially marked 5% of the quads as polyhedra and move one random vertex on the diagonal between 20% and 40%, as shown in the insets figures in blue. Note that some of the curves coincide, that is, Q_1 with Q_1 poly and Q_2 with Q_2 poly.

isoparametric elements, the choice of $\mathbf{g}$ is independent from the basis.

Quadrature. All integrals are computed numerically by means of quadrature points and weights, which translates the integrals into weighted sums. Although there are many strategies to generate quadrature data (e.g., Gaussian quadrature), all of them integrate exactly polynomials up to a given degree to ensure an appropriate approximation order. For instance, if we use one quadrature point in the element's center with weight 1, then we can integrate exactly constant functions.

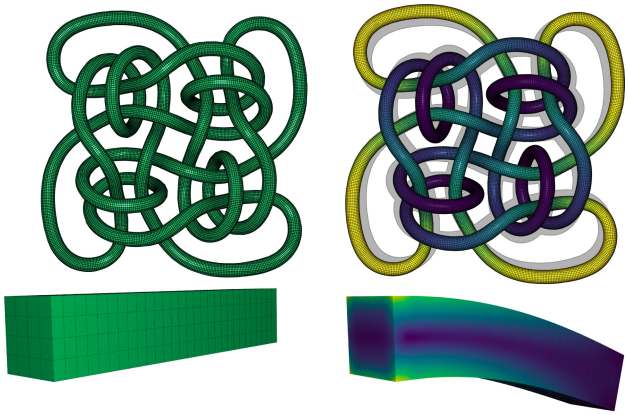


Fig. 23. Displacements computed solving linear elasticity on a pure hexahedral 3D model, using spline bases. Top: A complicated model with $\lambda = 1$ and $\mu = 1$. Bottom: A bended bar $\nu = 0.35$ and large young modulus $E = 2e5$.

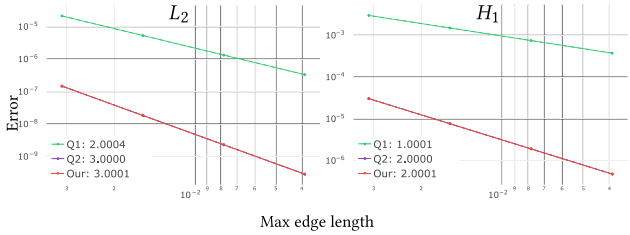


Fig. 24. Linear elasticity convergence plot in L_2 and H_1 norm on a regular grid in 2D.

Right-hand Side. The setup of the right-hand side $\mathbf{b}$ is done in a similar manner: Its entries are $b_i = \int_\Omega \phi_i(\mathbf{x}) f(\mathbf{x})\, d\mathbf{x}$.

Dirichlet boundary conditions are treated as constrained degrees of freedom. The Neumann boundary conditions are imposed by setting

$$b_j = \int_{\partial\Omega_N} \phi_j(\mathbf{x}) \cdot n(\mathbf{x})\, d\mathbf{x}$$

for any node j in $\partial\Omega_N$.

As for the stiffness matrix assembly, the basis and node construction for the right-hand side is performed locally.

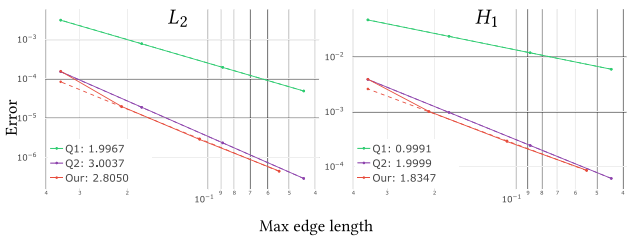


Fig. 25. Linear elasticity convergence plot in L_2 and H_1 norm on a hybrid mesh in 2D.

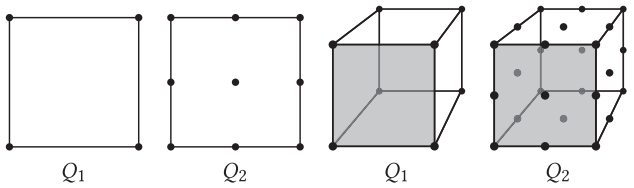


Fig. 26. Node position for the linear and quadratic bases in two and three dimensions.

B TEST FUNCTIONS

In our experiments, we use the test functions proposed in Franke (1979). For 2D:

$$f_{2D}(x_1,x_2) = \frac{3}{4}e^{-\frac{(9x_1-2)^2+(9x_2-2)^2}{4}} + \frac{3}{4}e^{-\frac{(9x_1+1)^2}{49}-\frac{9x_2+1}{10}} + \frac{1}{2}e^{-\frac{(9x_1-7)^2+(9x_2-3)^2}{4}} - \frac{1}{5}e^{-(9x_1-4)^2-(9x_2-7)^2},$$

and 3D:

$$f_{3D}(x_1,x_2,x_3) = \frac{3}{4}e^{-\frac{(9x_1-2)^2+(9x_2-2)^2+(9x_3-2)^2}{4}} + \frac{3}{4}e^{-\frac{(9x_1+1)^2}{49}-\frac{9x_2+1}{10}-\frac{9x_3+1}{10}} + \frac{1}{2}e^{-\frac{(9x_1-7)^2+(9x_2-3)^2+(9x_3-5)^2}{4}} - \frac{1}{5}e^{-(9x_1-4)^2-(9x_2-7)^2-(9x_3-5)^2}.$$

C BASIS FUNCTIONS

We use Lagrange tensor product function to interpolate between the nodes in quadrilateral and hexahedral elements. We provide the explicit formulation for Q_1 and Q_2 in 2D, and the 3D formulation follows. The four linear bases are constructed from the 1D linear bases

$$\alpha_1(t) = 1 - t \qquad \text{and} \qquad \alpha_2(t) = t$$

as the tensor products

$$\phi_1(u,v) = \alpha_1(u)\,\alpha_1(v), \qquad \phi_2(u,v) = \alpha_1(u)\,\alpha_2(v),$$
$$\phi_3(u,v) = \alpha_2(u)\,\alpha_1(v), \qquad \phi_4(u,v) = \alpha_2(u)\,\alpha_2(v).$$

Similarly, the nine quadratic bases follow from the three quadratic polynomials,

$$\theta_1(t) = (1-t)\,(1-2t), \quad \theta_2(t) = 4t\,(1-t), \quad \theta_3(t) = t\,(2t-1)$$

as

$$\phi_1(u,v) = \theta_1(u)\,\theta_1(v), \quad \phi_2(u,v) = \theta_1(u)\,\theta_2(v), \quad \phi_3(u,v) = \theta_1(u)\,\theta_3(v),$$
$$\phi_4(u,v) = \theta_2(u)\,\theta_1(v), \quad \phi_5(u,v) = \theta_2(u)\,\theta_2(v), \quad \phi_6(u,v) = \theta_2(u)\,\theta_3(v),$$
$$\phi_7(u,v) = \theta_3(u)\,\theta_1(v), \quad \phi_8(u,v) = \theta_3(u)\,\theta_2(v), \quad \phi_9(u,v) = \theta_3(u)\,\theta_3(v).$$

D POLYHEDRAL BASIS CONSTRAINTS

We restrict the detailed explanation to 2D, and the three-dimensional case follows. Let $\mathbf{p}_i = (x_i, y_i)$, $i = 1,\dots,s$, be the set of collocation points, that is, the points where we know the function values. For the FEM basis ϕ_j that is nonzero on the polyhedral element P, we want to solve the least-squares system $\mathbf{Aw} = \mathbf{b}$, where

$$\mathbf{A} = \begin{pmatrix} \psi_1(\mathbf{p}_1) & \dots & \psi_k(\mathbf{p}_1) & 1 & x_1 & y_1 & x_1\,y_1 & x_1^2 & y_1^2 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \psi_1(\mathbf{p}_s) & \dots & \psi_k(\mathbf{p}_s) & 1 & x_s & y_s & x_s\,y_s & x_s^2 & y_s^2 \end{pmatrix},$$

$\mathbf{b}$ is the evaluation of the basis ϕ_j from the neighbouring elements on the on the collocation points, and $\mathbf{w} = (w_1,\dots,w_k, a_{00}, a_{10}, a_{01}, a_{11}, a_{20}, a_{02})$.

Now, to ensure consistency, we need that

$$-\int_{\mathrm{g}(\widehat{\mathcal{M}})} \Delta q\phi_j = \int_{\mathrm{g}(\widehat{\mathcal{M}})} \nabla q \cdot \nabla\phi_j$$

holds for any of the five monomials. We now split the previous integral over the polygon P and over the known non-polygonal part $\overline{P} = \mathrm{g}(\widehat{\mathcal{M}}) \setminus P$,

$$\int_P \Delta q\phi_j + \int_P \nabla q \cdot \nabla\phi_j = -\int_{\overline{P}} \Delta q\phi_j - \int_{\overline{P}} \nabla q \cdot \nabla\phi_j.$$

We remark that the right-hand side of this equation is known since the bases on $\overline{P}$ are given, and we call the five term c_{ij} following the same indices as a_{ij} (e.g., $c_{20} = -\int_{\overline{P}} \Delta x^2\phi_j - \int_{\overline{P}} \nabla x^2 \cdot \nabla\phi_j$).

We now evaluate the left-hand side for the five 2D monomials

$$\int_P \Delta x\phi_j + \int_P \nabla x \cdot \nabla\phi_j = \int_P \frac{\partial\phi_j}{\partial x},$$
$$\int_P \Delta y\phi_j + \int_P \nabla x \cdot \nabla\phi_j = \int_P \frac{\partial\phi_j}{\partial y},$$
$$\int_P \Delta(xy)\phi_j + \int_P \nabla(xy) \cdot \nabla\phi_j = \int_P y\frac{\partial\phi_j}{\partial x} + \int_P x\frac{\partial\phi_j}{\partial y},$$
$$\int_P \Delta x^2\phi_j + \int_P \nabla x^2 \cdot \nabla\phi_j = \int_P 2\phi_j + \int_P 2x\frac{\partial\phi_j}{\partial x},$$
$$\int_P \Delta y^2\phi_j + \int_P \nabla y^2 \cdot \nabla\phi_j = \int_P 2\phi_j + \int_P 2y\frac{\partial\phi_j}{\partial y}.$$

By plugging the definition of ϕ_j over P, we obtain the following consistency constraints for the coefficients $a_{00}, a_{10}, a_{01}, a_{11}, a_{20}, a_{02}$:

$$\sum_{i=1}^{k} w_i^j \int_P \frac{\partial\psi}{\partial x} + a_{10}|P| + a_{11}\int_P y + 2a_{20}\int_P x = c_{10}$$
$$\sum_{i=1}^{k} w_i^j \int_P \frac{\partial\psi}{\partial y} + a_{01}|P| + a_{11}\int_P x + 2a_{02}\int_P y = c_{01},$$
$$\sum_{i=1}^{k} w_i^j \int_P \left(y\frac{\partial\psi}{\partial x} + x\frac{\partial\psi}{\partial y}\right) + a_{10}\int_P x + a_{01}\int_P y + a_{11}$$

$$\times \int_P x^2 + y^2 + 2(a_{20} + a_{02}) \int_P xy = c_{11}$$

$$2\sum_{i=1}^{k} w_i^j \int_P \left(\psi + x\frac{\partial \psi}{\partial x}\right) + 2a_{00} + 4a_{10}\int_P x + 2a_{01}\int_P y + 4a_{11}\int_P xy$$

$$+6a_{20}\int_P x^2 + 2a_{02}\int y^2 = c_{20},\ 2\sum_{i=1}^{k} w_i^j \int_P \left(\psi + y\frac{\partial \psi}{\partial y}\right) + 2a_{00}$$

$$+ 2a_{10}\int_P x + 4a_{01}\int_P y + 4a_{11}\int_P xy + 2a_{20}\int_P x^2 + 6a_{02}\int y^2 = c_{02}.$$

REFERENCES

Martin Aigner, Christoph Heinrich, Bert Jüttler, Elisabeth Pilgerstorfer, Bernd Simeon, and Vuong. 2009. *Swept Volume Parameterization for Isogeometric Analysis.*

Douglas Arnold, Daniele Boffi, and Richard Falk. 2002. Approximation by quadrilateral finite elements. *Math. Comput.* (2002).

Yuri Bazilevs, L. Beirao da Veiga, J Austin Cottrell, Thomas J. R. Hughes, and Giancarlo Sangalli. 2006. Isogeometric analysis: Approximation, stability and error estimates for h-refined meshes. *Math. Meth. Appl. Sci.* (2006).

L. Beirão Da Veiga, F. Brezzi, A. Cangiani, G. Manzini, L. D. Marini, and A. Russo. 2013. Basic principles of virtual element methods. *Math. Meth. Appl. Sci.* (2013).

Steven E. Benzley, Ernest Perry, Karl Merkley, Brett Clark, and Greg Sjaardema. 1995. A comparison of all hexagonal and all tetrahedral finite element meshes for elastic and elasto-plastic analysis. In *Proceedings of the 4th International Meshing Roundtable.*

J. E. Bishop. 2014. A displacement-based finite element formulation for general polyhedra using harmonic shape functions. *Int. J. Numer. Methods Eng.* (2014).

Dietrich Braess. 2007. *Finite Elements: Theory, Fast Solvers, and Applications in Solid Mechanics.*

R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. 2018. Neural ordinary differential equations. *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.). Curran Associates, Inc., 6571–6583. http://papers.nips.cc/paper/7892-neural-ordinary-differential-equations.pdf.

A. O. Cifuentes and A. Kalbag. 1992. A performance study of tetrahedral and hexahedral elements in 3-D finite element structural analysis. *Finite Elements in Analysis and Design* (1992).

J. Austin Cottrell, Thomas J. R. Hughes, and Yuri Bazilevs. 2009. *Isogeometric Analysis: Toward Integration of CAD and FEA.*

L. Beirão da Veiga, A. Buffa, D. Cho, and G. Sangalli. 2011. Isogeometric analysis using T-splines on two-patch geometries. *Comput. Meth. Appl. Mech. Eng.* (2011).

Blanca Ayuso de Dios, Konstantin Lipnikov, and Gianmarco Manzini. 2016. The nonconforming virtual element method. *ESAIM: Mathematical Modelling and Numerical Analysis* (2016).

Luke Engvall and John A. Evans. 2017. Isogeometric unstructured tetrahedral and mixed-element Bernstein-Bezier discretizations. *Comput. Meth. Appl. Mech. Eng.* (2017).

Xianzhong Fang, Weiwei Xu, Hujun Bao, and Jin Huang. 2016. All-hex meshing using closed-form induced polycube. *ACM Trans. Graph.* (2016).

Michael S. Floater, Géza Kós, and Martin Reimers. 2005. Mean value coordinates in 3D. *Comput. Aided Geom. Des.* 22, 7 (2005), 623–631. https://doi.org/10.1016/j.cagd.2005.06.004

Richard Franke. 1979. A Critical Comparison of Some Methods for Interpolation of Scattered Data.

Xiaoming Fu, Chongyang Bai, and Yang Liu. 2016. Efficient volumetric PolyCube-map construction. *Comput. Graph. Forum* (2016).

Xifeng Gao, Wenzel Jakob, Marco Tarini, and Daniele Panozzo. 2017a. Robust hex-dominant mesh generation using field-guided polyhedral agglomeration. *ACM Trans. Graph.* 36, 4 (2017), 1–13. https://doi.org/10.1145/3072959.3073676

Xifeng Gao, Daniele Panozzo, Wenping Wang, Zhigang Deng, and Guoning Chen. 2017b. Robust structure simplification for hex re-meshing. *ACM Trans. Graph.* 36, 6 (2017), 1–13. https://doi.org/10.1145/3130800.3130848

James Gregson, Alla Sheffer, and Eugene Zhang. 2011. All-hex mesh generation via volumetric PolyCube deformation. *Comput. Graph. Forum* (2011).

Kai Hormann and Natarajan Sukumar. 2008. Maximum entropy coordinates for arbitrary polytopes. *Comput. Graph. Forum* 27, 5 (2008), 1513–1520. https://doi.org/10.1111/j.1467-8659.2008.01292.x

Jin Huang, Tengfei Jiang, Zeyun Shi, Yiying Tong, Hujun Bao, and Mathieu Desbrun. 2014. L1-based construction of polycube maps from complex shapes. *ACM Trans. Graph.* (2014).

Jin Huang, Yiying Tong, Hongyu Wei, and Hujun Bao. 2011. Boundary aligned smooth 3D cross-frame field. *ACM Trans. Graph.* (2011).

Thomas J. R. Hughes, John A. Cottrell, and Yuri Bazilevs. 2005. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Comput. Meth. Appl. Mech. Eng.* (2005).

Thomas J. R. Hughes. 2000. *The Finite Element Method. Linear Static and Dynamic Finite Element Analysis.*

Yasushi Ito, Alan M. Shih, and Bharat K. Soni. 2009. Octree-based reasonable-quality hexahedral mesh generation using a new set of refinement templates. *Int. J. Numer. Methods Eng.* (2009).

Tengfei Jiang, Jin Huang, Yiying Tong Yuanzhen Wang, and Hujun Bao. 2014. Frame field singularity correction for automatic hexahedralization. *IEEE Trans. Vis. Comput. Graph.* (2014).

Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. 2007. Harmonic coordinates for character articulation. *ACM Trans. Graph.* 26, 3 (2007), 71. https://doi.org/10.1145/1276377.1276466

Tao Ju, Scott Schaefer, and Joe Warren. 2005. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph.* 24, 3 (2005), 561. https://doi.org/10.1145/1073204.1073229

Ilya Kostrikov, Zhongshi Jiang, Daniele Panozzo, Denis Zorin, and Joan Bruna. 2018. Surface networks. In *Proceedings f the IEEE Conference on Computer Vision and Pattern Recognition (CVF'18).* IEEE, 2540–2548.

Na Lei, Xiaopeng Zheng, Jian Jiang, Yu-Yao Lin, and David Xianfeng Gu. 2017. Quadrilateral and hexahedral mesh generation based on surface foliation theory. *Comput. Meth. Appl. Mech. Eng.* (2017).

Bo Li, Xin Li, Kexiang Wang, and Hong Qin. 2013. Surface mesh to volumetric spline conversion with generalized polycubes. *IEEE Trans. Vis. Comput. Graph.* (2013).

Yufei Li, Yang Liu, Weiwei Xu, Wenping Wang, and Baining Guo. 2012. All-hex meshing using singularity-restricted field. *ACM Trans. Graph.* (2012).

Konstantin Lipnikov, Gianmarco Manzini, and Mikhail Shashkov. 2014. Mimetic finite difference method. *J. Comput. Phys.* (2014).

Marco Livesu, Nicholas Vining, Alla Sheffer, James Gregson, and Riccardo Scateni. 2013. PolyCut: Monotone graph-cuts for PolyCube base-complex construction. *ACM Trans. Graph.* (2013).

Gianmarco Manzini, Alessandro Russo, and N. Sukumar. 2014. New perspectives on polygonal and polyhedral finite element methods. *Math. Meth. Appl. Sci.* (2014).

Loïc Maréchal. 2009. Advances in octree-based all-hexahedral mesh generation: Handling sharp features. In *Proceedings of the 18th International Meshing Roundtable.* Springer, 65–84. https://link.springer.com/chapter/10.1007/978-3-642-04319-25

Sebastian Martin. 2011. Flexible, unified and directable methods for simulating deformable objects. (2011). Ph.D. dissertation, ETH Zurich.

Sebastian Martin, Peter Kaufmann, Mario Botsch, Martin Wicke, and Markus Gross. 2008. Polyhedral finite elements using harmonic basis functions. In *Proceedings of the Symposium on Geometry Processing.*

Tobias Martin and Elaine Cohen. 2010. Volumetric parameterization of complex objects by respecting multiple materials. *Comput. Graph.* (2010).

Matthias Nieser, Ulrich Reitebuch, and Konrad Polthier. 2011. CubeCover - Parameterization of 3D volumes. *Comput. Graph. Forum* (2011).

Steven J. Owen and Sunil Saigal. 2000. H-Morph: An indirect approach to advancing front hex meshing. *Int. J. Numer. Methods Eng.* (2000).

Julian Panetta, Qingnan Zhou, Luigi Malomo, Nico Pietroni, Paolo Cignoni, and Denis Zorin. 2015. Elastic textures for additive fabrication. *ACM Trans. Graph.* (2015).

Cosmin G. Petra, Olaf Schenk, and Mihai Anitescu. 2014a. Real-time stochastic optimization of complex energy systems on high-performance computers. *IEEE Computing in Science & Engineering* (2014).

Cosmin G. Petra, Olaf Schenk, Miles Lubin, and Klaus Gärtner. 2014b. An augmented incomplete factorization approach for computing the Schur complement in stochastic optimization. *SIAM J. Sci. Comput.* 36, 2 (2014), C139–C162.

Thomas W. Sederberg, David L. Cardon, G. Thomas Finnigan, Nicholas S. North, Jianmin Zheng, and Tom Lyche. 2004. T-spline simplification and local refinement. *ACM Trans. Graph.* (2004).

Jason F. Shepherd and Chris R. Johnson. 2008. Hexahedral mesh generation constraints. *Eng. Comput.* (2008).

Jonathan Richard Shewchuk. 1996. *Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator.*

Hang Si. 2015. TetGen, a delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw.* (2015).

Dmitry Sokolov, Nicolas Ray, Lionel Untereiner, and Bruno Lévy. 2016. Hexahedral-dominant meshing. *ACM Trans. Graph.* (2016).

Matthew L. Staten, Steven J. Owen, and Ted D. Blacker. 2005. *Unconstrained Paving & Plastering: A New Idea for All Hexahedral Mesh Generation.*

Marco Tarini, Kai Hormann, Paolo Cignoni, and Claudio Montani. 2004. PolyCube-maps. *ACM Trans. Graph.* (2004).

Deepesh Toshniwal, Hendrik Speleers, René R. Hiemstra, and Thomas J. R. Hughes. 2017. Multi-degree smooth polar splines: A framework for geometric modeling and isogeometric analysis. *Comput. Meth. Appl. Mech. Eng.* (2017).

Xiaodong Wei, Yongjie Jessica Zhang, Deepesh Toshniwal, Hendrik Speleers, Xin Li, Carla Manni, John A. Evans, and Thomas J. R. Hughes. 2018. Blended B-spline construction on unstructured quadrilateral and hexahedral meshes with optimal convergence rates in isogeometric analysis. *Comput. Meth. Appl. Mech. Eng.* (2018).

Soji Yamakawa and Kenji Shimada. 2003. Fully-automated hex-dominant mesh generation with directionality control via packing rectangular solid cells. *Int. J. Numer. Methods Eng.* (2003).

Y. J. Zhang, X. Liang, and Guoliang Xu. 2013. A robust 2-refinement algorithm in octree or rhombic dodecahedral tree based all-hexahedral mesh generation. *Comput. Meth. Appl. Mech. Eng.* (2013).

Received June 2018; revised January 2019; accepted February 2019