VEMPIC:
Particle-in-Polyhedron Fluid Simulation for Intricate Solid Boundaries

MICHAEL TAO, University of Toronto
CHRISTOPHER BATTY, University of Waterloo
MIRELA BEN-CHEN, Technion - Israel Institute of Technology
EUGENE FIUME, Simon Fraser University
DAVID I. W. LEVIN, University of Toronto

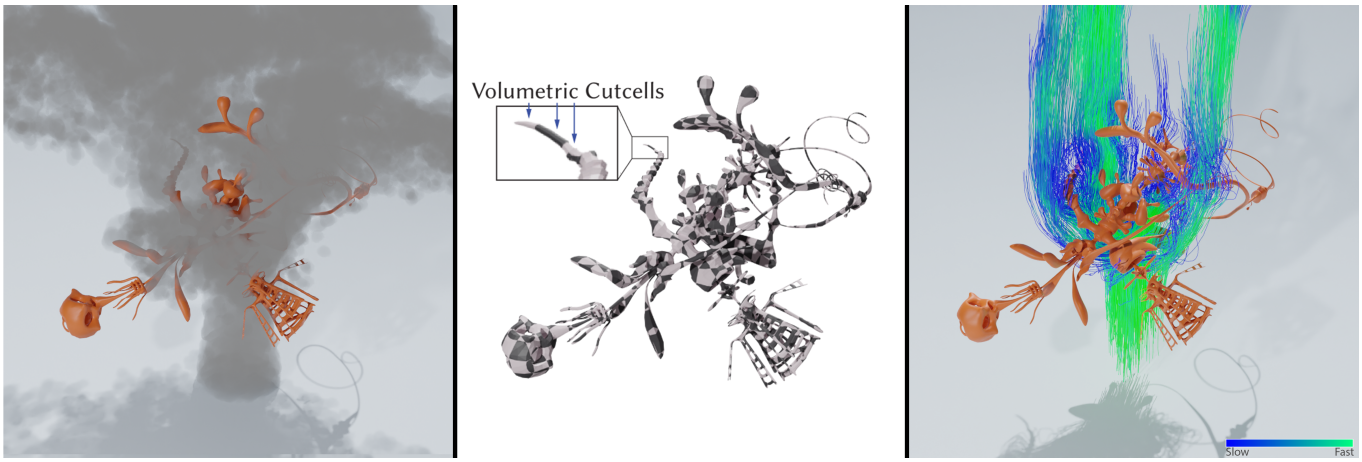


Fig. 1. Our method allows for fluid simulation around extremely complex geometry. The *YeahRight* mesh is challenging for fluid simulators due to its many thin features and complex topology. We are able to generate simulations that interact with fine geometric features better than previous approaches (left) because we leverage polyhedral cut-cells which can exactly represent the input triangle mesh (center). Even the smallest tendrils of the obstacle influence the flow of the particle trajectories (right).

The comprehensive visual modeling of fluid motion has historically been a challenging task, due in no small part to the difficulties inherent in geometries that are non-manifold, open, or thin. Modern geometric cut-cell mesh generators have been shown to produce, both robustly and quickly, workable volumetric elements in the presence of these problematic geometries, and the resulting volumetric representation would seem to offer an ideal infrastructure with which to perform fluid simulations. However, cut-cell mesh elements are general polyhedra that often contain holes and are non-convex; it is therefore difficult to construct the explicit function spaces required to employ standard functional discretizations, such as the Finite Element Method. The Virtual Element Method (VEM) has recently emerged as a functional discretization that successfully operates with complex polyhedral elements through a weak formulation of its function spaces. We present a novel cut-cell fluid simulation framework that exactly represents boundary geometry during the simulation. Our approach enables, for the first time, detailed fluid simulation with "in-the-wild" obstacles, including ones that contain non-manifold parts, self-intersections, and extremely thin features. Our key technical contribution is the generalization of the Particle-In-Cell fluid simulation methodology to arbitrary polyhedra using VEM. Coupled with a robust cut-cell generation scheme, this produces a fluid simulation algorithm that can operate on previously infeasible geometries without requiring any additional mesh modification or repair.

CCS Concepts: • **Computing methodologies** → **Physical simulation**.

Additional Key Words and Phrases: Computer Graphics, physically-based simulation, fluid simulation

ACM Reference Format:
Michael Tao, Christopher Batty, Mirela Ben-Chen, Eugene Fiume, and David I. W. Levin. 2022. VEMPIC: Particle-in-Polyhedron Fluid Simulation for Intricate Solid Boundaries. *ACM Trans. Graph.* 41, 4, Article 115 (July 2022), 22 pages. https://doi.org/10.1145/3528223.3530138

Authors' addresses: Michael Tao, University of Toronto, mtao@dgp.toronto.edu; Christopher Batty, University of Waterloo, christopher.batty@uwaterloo.ca; Mirela Ben-Chen, Technion - Israel Institute of Technology, mirelacs.technion.ac.il; Eugene Fiume, Simon Fraser University, efiume@sfu.edu; David I. W. Levin, University of Toronto, diwlevin@cs.toronto.edu.

0730-0301/2022/7-ART115
https://doi.org/10.1145/3528223.3530138

1 INTRODUCTION

The desire to numerically reproduce the diverse behaviors of incompressible fluid phenomena presents several daunting challenges. Incompressibility introduces a global structure that ensures local deformations affect the entire fluid volume instantaneously. At the

same time, Kolmogorov's Theory of Turbulence states that the energy spectrum of a fluid cascades towards ever-higher frequencies [Frisch and Kolmogorov 1995], generating increasingly intricate details. Practical applications further demand the ability to faithfully incorporate solid obstacles that can possess significant geometric complexity, including non-manifold, high genus, open, or thin components. Our aim in this work is to approach these challenges in a holistic yet flexible fashion through a novel synthesis and generalization of recent numerical techniques: *Cut-Cell Meshes*, the *Virtual Element Method* (VEM), and *Particle-In-Cell* (PIC) schemes.

Beginning with spatial discretization in the presence of complex boundaries, we use cut-cell meshes [Tao et al. 2019], which are formed by the intersection of a Cartesian grid with a standard triangle mesh boundary description. Cut-cell meshes provide a natural scaffolding with which to solve the equations necessary for global incompressibility, while conforming to the original boundary shape, even in the difficult geometric cases alluded to above.

Cut-cell meshes produced from nontrivial boundary geometry inevitably possess mesh elements whose shapes strain the capabilities of typical finite volume/element schemes. We therefore pair cut-cell meshes with the recently developed Virtual Element Method [Beirão da Veiga et al. 2013, 2014], a powerful extension of finite element concepts that was developed specifically to support meshes whose elements are arbitrary polygons/polyhedra while retaining support for high-order function spaces.

Finally, while VEM affords us an effective Eulerian solution for incompressibility enforcement, experience has shown that Lagrangian particle-based treatments of advection can often better preserve fine-scale vorticial structure. As a result, a range of hybrid Particle-In-Cell (PIC) methods, which transfer information between mesh and particle representations, have been proposed to bridge the gap between the Eulerian and Lagrangian perspectives [Zhu and Bridson 2005; Jiang et al. 2015]. Prior methods in the PIC family either assume a uniform Cartesian grid and considerably simplify the domain geometry, leading to lower quality boundary interactions, or have not considered the significant level of obstacle complexity studied in our work [Azevedo et al. 2016; Edwards and Bridson 2014]. We exploit the mathematical ideas underpinning VEM to develop a tailored PIC scheme that interacts seamlessly with, and leverages the structure of, our geometry-conforming cut-cell mesh. Our resulting hybrid VEMPIC simulator is able to produce detailed incompressible fluid flows in the presence of extremely intricate geometries on which prior methods would typically struggle or fail outright.

2 RELATED WORK

Fluid simulators can be categorized based on whether their degrees of freedom (DOFs) move over time (Lagrangian) or remain fixed in space (Eulerian). The most common purely Lagrangian techniques use a collection of disconnected particles, each denoting the center of a small distribution of fluid material [Desbrun and Gascuel 1996; Monaghan 1992]. In recent years Lagrangian techniques based on dynamic meshes have been developed, such as moving tetrahedral meshes [Clausen et al. 2013; Misztal et al. 2013] or Voronoi/power diagrams of Lagrangian point clouds [Sin et al. 2009; De Goes et al. 2015]. Semi-Lagrangian advection methods [Stam 1999] which combine ideas from both the Eulerian and Lagrangian perspectives are popular in computer graphics, particularly for smoke simulation, due to their unconditional stability [Nielsen et al. 2018]. Although this stability initially came at the cost of overly smooth results, there have been various efforts to reduce these effects [Fedkiw et al. 2001; Selle et al. 2008]. Finally, in contrast to semi-Lagrangian schemes, traditional *purely* Eulerian discretizations of the advection terms are comparatively rare [Foster and Metaxas 1996; Mullen et al. 2009]. For many fluid animation problems hybrid particle/grid schemes are highly effective; we discuss these below.

2.1 Particle + Mesh Schemes

The FLuid-Implicit-Particle (FLIP) method [Zhu and Bridson 2005] is an excellent exemplar of modern hybrid "particle + mesh" (i.e., particle-in-cell) fluid simulators. Such methods alternate between the two representations, allowing particles to naturally handle (Lagrangian) advection of fluid material and the mesh to handle (Eulerian) incompressibility enforcement. In particular, FLIP substantially mitigates the smoothing induced by repeated grid-to-particle-to-grid interpolations by transferring only the change in grid data; i.e., propagating particle data increments, rather than overwriting their data as the earliest particle-in-cell (PIC) approaches did. Further advances have shown that the "overwriting" strategy can still be effective if particles are augmented with linear or polynomial, rather than constant, models of the local velocity, as in the Affine PIC (APIC) [Jiang et al. 2015] and Polynomial PIC (PolyPIC) [Fu et al. 2017] methods. Several techniques have sought to improve the cost and/or level of detail captured by FLIP, by only storing particles near the surface [Ferstl et al. 2016], using adaptive particle sizes [Ando et al. 2013], or by resolving sub-grid dynamics via auxiliary processes [Mercier et al. 2015; Pfaff et al. 2012; Kim et al. 2008; Bojsen-Hansen and Wojtan 2013]. Edwards and Bridson extended PIC to high order [Edwards and Bridson 2012] using higher order transfer operators, but considered only periodic domains without obstacles.

Sulsky et al. first proposed the material point method (MPM) as a generalization of classical PIC/FLIP schemes from fluid dynamics [Brackbill et al. 1988] to support solid dynamics [Sulsky et al. 1994]. It is especially effective for materials with complex constitutive behavior, such as snow [Stomakhin et al. 2013], granular media [Daviet and Bertails-Descoubes 2016], or viscoplastic phenomena [Ram et al. 2015], among others. Hu et al. [2018] developed a variant of MPM to handle thin solids (i.e., with discontinuous velocity fields) based on a color-field approach.

Unlike earlier PIC-based methods that used trilinear interpolation of velocity, recent MPM methods use tensor product B-splines [Stomakhin et al. 2013]. This practice takes advantage of the regularity of Cartesian grids by using velocity data from neighboring cells to improve the smoothness of the velocity field without additional per-cell velocity samples. Gagniere et al. [2020] use multiquadratic B-spline interpolation for velocity and trilinear interpolation within a fluid simulator based on cut-cell finite elements and implicit semi-Lagrangian advection.

Recently, Fang et al. [2020] proposed a particle-based interface quadrature treatment to handle free slip boundary conditions in Cartesian grid MPM for two-way fluid-solid coupling, sidestepping the need for a conforming mesh. The objects that this approach supports remain fairly coarse relative to the MPM grid resolution. By contrast, our work extends PIC to support fine-scale objects with coarse grids using a combination of cut-cell meshes and polynomial function spaces.

2.2 Cut-cell Meshes

Tetrahedral meshing has long been applied to conform simulation elements with boundary geometry, including more recently for dynamic liquid surfaces (e.g., [Misztal et al. 2013; Clausen et al. 2013]). However, regular Cartesian grids have remained popular due to their simplicity, efficient construction, and ease of use, despite their difficulties in resolving complex boundaries. Cut-cell meshes have emerged as a popular extension of Cartesian grids to support non-axis-aligned geometry. We can distinguish two categories: implicit variants, which typically use per-node level set data to *approximate* the geometry contained in a cell, and explicit variants, which construct the *exact* cut-cell geometry directly.

Implicit cut-cells are easily constructed and enable high fidelity simulations when the geometry is well-resolved by the chosen grid resolution, but are topologically limited in a fundamental way: each grid cell can contain only one cut-cell and the adjacency graph between cut-cells must be a subset of that of the Cartesian grid. (With additional geometric and topological bookkeeping, this could theoretically be generalized to at most five cut-cells per grid cell in accordance with the corresponding marching cubes cases [Lorensen and Cline 1987], though it has not yet been done.) In practice, this limitation precludes domains with thin boundaries because fluid on both sides of the boundary will frequently belong to the same cell. Leakage between cells, as discussed by Guendelman et al. [2005] and embraced in the work of Fei et al. [2018], allows fluid material to pass through thin materials. The limitations above do not preclude multimaterial simulations, where the material on both sides of the cut-cell interface share similar discretizations, such as liquid-air [Boyd and Bridson 2012], fluid-elastic [Teng et al. 2016], and fluid-solid [Zarifi and Batty 2017] interactions; the resulting simulations can convincingly reproduce the intended phenomena.

On the other hand, explicit (or geometric) cut-cell meshes are generated by computing the intersection of each cell in a Cartesian mesh with the boundary geometry, typically given as a triangle mesh [Aftosmis et al. 1998]. The recently introduced *Mandoline* cut-cell mesher supports creating a volume discretization from open or even non-manifold geometry [Tao et al. 2019].

The Finite Element Method has been extended to support certain cut-cell mesh variants. The eXtended FEM [Moës et al. 1999] enriches the FEM function space with jump functions, and has been applied to cutting of shells and solids [Kaufmann et al. 2009b; Koschier et al. 2017]. Sifakis et al. [2007a] uses copies of the uncut elements' basis functions, in a virtual node approach for elastic objects [Molino et al. 2004], with an extension to support collisions and more flexible cutting [Sifakis et al. 2007b]. In the fluid context, Edwards and Bridson [2014] project their cut-cell geometry into a polygonal basis and define a projection scheme from particles to degrees of freedom, similar in spirit to our work. Their approach differs in the use of the Local Discontinuous Galerkin method, which adopts polynomial coefficients as the degrees of freedom with a Lax-Friedrichs condition to conform between cells, requiring more machinery to introduce boundary conditions. They use a regularized least-squares procedure to map particle velocities to polynomial coefficients, whereas our approach only requires integration with an unstructured quadrature rule, which can be computed as an unweighted average over sample points. Their work considered only simple volumetric obstacles, but emphasized the benefits of p-adaptivity for liquids.

More recently, the work of Azevedo et al. [2016] extended FLIP to geometric cut-cells by updating the cell adjacency graph for pressure projection and using spherical barycentric coordinates to define a (mostly) cell-conforming interpolation scheme for advection. The low order cell-centered pressure projection is essentially a modified finite volume or Discrete Exterior Calculus discretization [Hirani 2003]. This work was further extended by Chen et al. [2020] to handle free surfaces by assigning each cell multiple virtual pressure samples but only a single real pressure degree of freedom per cell. Tao et al. [2019] demonstrated that Mandoline yields meshes that are compatible with Azevedo's Poisson discretization. However, these methods depend on solving a Poisson problem with a graph-based-Laplacian on a dual mesh, using only one DOF per cell, so they are limited in the complexity of sub-cell flow that can be faithfully represented. Moreover, for advection, an additional reconstruction and interpolation mechanism is required to generate plausible boundary-respecting interior flows. Our use of a VEM discretization leverages the efficacy of higher order pressure solutions for navigating the fluid around complex contours; that is, the pressure projection itself is made more "geometry-aware", and the same polynomial spaces are used for both projection and advection.

Hyde and Fedkiw [2019] developed a different approach to approximately resolving fine sub-grid features on the interior of cells, where the motion of sub-grid obstacles is guaranteed to contribute a certain proportion of flux to cell boundaries according to the proportion of the cell they occupy. The method builds on staggered grid Laplacians, which are a particular instance of graph-based Laplacians. From the perspective of graphs, this method extends the standard finite difference graph by giving each obstacle its own node, and connecting those obstacle-nodes to cells adjacent to the obstacle. These additional degrees of freedom and connections bypass the standard limitations of graph-based Laplacians for performing pressure projection. The concurrent work of Lyu et al. [2021] similarly adopts a denser connectivity graph in the context of Lattice Boltzmann schemes to reduce aliasing along partially cut edges and to better handle flows near thin features. They resolve sub-voxel features by first projecting their nodes to thin feature boundaries and then removing all nodes within the voxel. Both of these methods make significant approximations of the real input geometry.

Unlike previous methods, we make the geometry a first-class citizen by using the explicit cut cell mesh to place degrees of freedom along the boundary in a precise fashion. We implement quadratic pressures to maintain incompressibility not only between cells but

in their interiors as well, without any complex notions of node placement or smearing of interior features onto cell boundaries. Placing degrees of freedom along all boundaries, including the boundaries of occluding geometry, gives our pressure system awareness of thin sub-grid features, allowing our method to easily resolve models with fine details such as the tendrils of the *YeahRight* example of Figure 1.

2.3 Polyhedral Discretization Methods

A principal difficulty in generalizing FEM to polyhedral domains is the construction of appropriate shape functions. Interpolation schemes for polyhedra such as harmonic coordinates are promising [Joshi et al. 2007], but constructing stiffness matrices for general geometry can be quite difficult. Techniques like the Method of Fundamental Solutions [Martin et al. 2008] can be used to estimate boundary conditions, but require care to define boundary data and still need an approximation via quadratures to evaluate stiffness matrices.

In the context of shape function choice, Discontinuous Galerkin methods relax the boundary conditions required by their shape functions so they can choose easily refined spaces of functions, such as polynomials. Some choose their shape function space as monomials in the ambient space [Kaufmann et al. 2009a], while others choose to define their polynomials on best fit tetrahedra [Edwards and Bridson 2014]. Similarly, Trusty et al. [2021] apply shape matching to match their deformations to a polynomial basis.

The Mimetic Finite Difference (MFD) method [Brezzi et al. 2009] moves away from shape functions by instead using finite difference/volume concepts while preserving discretized versions of Stokes' and Green's Theorems. MFD defines graph-based discrete differential operators and introduces metric data such as length and volume via adjoint operators. This strategy has connections to Discrete Exterior Calculus methods which have been widely used in geometry processing and elsewhere [Mullen et al. 2009; De Goes et al. 2020].

Merging MFD concepts with those of FEM, the Virtual Element Method [Beirão da Veiga et al. 2013] further decouples the mesh degrees of freedom from the interior functional representation by only guaranteeing a smooth notion of integration-by-parts. A *virtual* function space with basis functions supported on cell interiors and boundaries encapsulates the geometric details in functional form. These functions, however, cannot be explicitly evaluated at points and are instead defined via inner products, while an auxiliary (polynomial) function space is used for evaluating functions at points. The necessary inner products are defined to guarantee integration-by-parts. As we will show in Section 5.3, a combination of analytically integrating polynomials over polyhedra and applying a projection from virtual functions to polynomial functions suffices to determine stiffness matrices in the virtual space.

3 FLUID DISCRETIZATION

We seek to simulate inviscid fluid flow as described by the incompressible Euler equations,

$$\rho \frac{\partial \mathbf{u}}{\partial t} = -\rho \mathbf{u} \cdot \nabla \mathbf{u} - \nabla p + \mathbf{f}, \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{2}$$

with vector-valued velocity field $\mathbf{u}$, scalar density field ρ, vector-valued external forces $\mathbf{f}$, pressure field p, and time t.

3.1 PIC

We discretize these equations using a general particle-in-cell (PIC) paradigm, maintaining both particle and mesh representations of the fluid velocities [Zhu and Bridson 2005]. The dual representation leverages classical Chorin splitting [Chorin 1968], which separates the Euler equations into alternating momentum updates and incompressibility-enforcing pressure projections:

$$\tilde{\mathbf{u}}_{j+1} = \mathbf{u}_j + h\mathbf{f}, \tag{3}$$

$$p_{j+1} = \operatorname*{argmin}_p \|\tilde{\mathbf{u}}_{j+1} - \nabla p\|^2, \tag{4}$$

$$\hat{\mathbf{u}}_{j+1} = \tilde{\mathbf{u}}_{j+1} - \nabla p, \tag{5}$$

$$\mathbf{u}_{j+1} = \hat{\mathbf{u}}_{j+1} + h(\hat{\mathbf{u}}_{j+1} \cdot \nabla)\hat{\mathbf{u}}_{j+1}, \tag{6}$$

where, j is the index of the current simulation step and h is the timestep size. Equation (3) applies external forces, Equations (4) and (5) apply a variational formulation of pressure projection, and Equation (6) advects fluid material forward in space. This variational formulation of pressure projection adopted above is described by Batty et al. [2007].

A typical PIC implementation is given in Algorithm 1, using particles to handle advection and a Cartesian grid to handle pressure projection. Because advection models the evolution of velocity due to the fluid's motion, moving particles with their velocities forms an efficient discretization of the equation. Representing the velocity field redundantly, with both particles and a grid, simultaneously enables energetic motion and high-fidelity incompressibility in exchange for adding particle-to-grid and grid-to-particle transfers that synchronize the two representations. The transfers are implemented by sampling the velocity fields represented by the particles or the Cartesian grid. Various options are available for these transfers, such as trilinear or higher order interpolation and weighting [Zhu and Bridson 2005], radial basis functions [Batty and Bridson 2008; Ando et al. 2013], or tensor product B-spline kernels [Stomakhin et al. 2013].

Algorithm 1: Basic PIC-based method

```
Function simulationStep(stepsize)
    particleToMesh();                  // Section 5.1
    applyForces(stepsize);
    pressureProjection();              // Section 5.3
    meshToParticle();                  // Section 5.2
    advect(stepsize);                  // Section 6.3
end
```

To combine PIC with VEM, we consider *three* distinct representations: particle data, discrete mesh data, and the polynomials used to represent the continuous velocity field within each cell. The discrete mesh data values are stored on cell boundaries and interiors, and thus their number depends directly on the complexity of the cell geometry; by contrast, the polynomial representation is composed of per-cell function spaces that do not. This separation of mesh

data from polynomial coefficients is what allows VEM to operate on meshes with complex element geometries like our polyhedral cut-cells.

Our pipeline (Figure 2) follows the same general schema as a standard PIC simulator, but we replace the Cartesian grid with a cut-cell mesh, and replace both the finite difference Laplacian and trilinear interpolation used in standard PIC with VEM tools. More precisely, we use particles to evaluate the velocity at the faces of each cut-cell (Section 5.1), use a quadratic VEM scheme to perform a cut-cell pressure projection (Section 5.3), and lastly use the resulting per-cell linear velocity fields both to resample the particle velocity values (Section 5.2) and to advect the fluid particles (Section 6.3).

4 VIRTUAL ELEMENT METHOD PRELIMINARIES

The Virtual Element Method (VEM) enables the solution of partial differential equations on meshes composed of general polyhedral elements. Like the Finite Element Method, VEM approximates functions inside elements as a linear combination of basis functions,

$$f \approx \sum_i m_i \phi_i, \tag{7}$$

where f is the function to be approximated, m_i are the VEM coefficients (i.e., degrees of freedom) stored at points on the mesh, and ϕ_i are the basis functions. As is typical in many FEM formulations, VEM assumes that the shape functions ϕ_i are smooth and have the Kronecker Delta property. However, unlike the explicit basis functions of standard FEM, VEM assumes that the values of its basis functions are only known at the boundary of each polyhedral element and at a sparse set of points inside its volume. Because these basis functions are not known everywhere they are called *Virtual*; we denote their space by $\mathcal{V}$, and whenever possible index associated quantities by i. Table 1 summarizes this and other notation used throughout the paper.

To enable computation of derivatives and integrals of these virtual functions we will define a projection onto the space of polynomials, $\mathcal{P}$, and perform such mathematical operations there instead. (We will typically index $\mathcal{P}$ quantities by j.) This projection operator $\Pi : \mathcal{V} \to \mathcal{P}$ can be constructed from sparse virtual function information via Stokes' theorem (§5.3). Importantly, this construction relies on the assumption that each degree of freedom, m_i, can be computed via an inner product of the form

$$m_i(f) = \langle \phi_i, f \rangle = \frac{1}{|c_i|} \int_{c_i} f g_i, \tag{8}$$

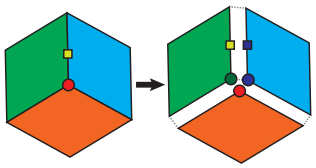
where c_i is a domain of integration (e.g., cell or face), and $|c_i|$ denotes the area/volume of c_i. Conveniently, the so-far-undefined functions, g_i, are themselves simply polynomials whose definition depends on the $\mathcal{P}$ we are projecting our virtual functions onto, and are known a priori (§5.3.2). With these notions in hand, one can define a convergent and consistent virtual element discretization.

4.1 Conformal vs. Non-conformal VEM

In our formulation, we follow "non-conformal" VEM [de Dios et al. 2016], which uses integrated DOFs. This choice contrasts with the original "conformal" VEM approach [Beirão da Veiga et al. 2013], emphasized in standard introductions [Beirão da Veiga et al. 2014; Mengolini et al. 2019], which uses pointwise DOFs. Figure 3 highlights the differences in how these two VEM variations store data.

Table 1. Notation

Symbol	Definition
$\mathcal{V}$	Virtual function space
$\mathcal{P}$	Polynomial function space
ϕ_i	Virtual basis function
ψ_j	Polynomial basis function
m_i	Measurement functor for ϕ_i
c_i	Integration domain for m_i
g_i	Integration weights for m_i
Π	VEM projection operator
$\mathbf{G}$	Green's Matrix
$\mathbf{L}^{\mathcal{V}}$	$\mathcal{V}$ Laplacian
$\mathbf{L}^{\mathcal{P}}$	$\mathcal{P}$ Laplacian
$\hat{\mathbf{L}}^{\mathcal{P}}$	Regularized $\mathcal{P}$ Laplacian
$\mathbf{L}$	VEM Laplacian $\approx \mathbf{L}^{\mathcal{V}}$
ξ	Integration by parts cell coeffs
ξ^{∇}	Integration by parts boundary coeffs



There are several advantages to our use of non-conformal VEM, primarily relating to ease of implementation. In particular, supporting thin and non-manifold subcell geometry requires duplicating geometry and the DOFs that lie on those geometries (see inset 2D illustration). However, even in the presence of non-manifoldness, mesh faces have exactly two sides; thus, because non-conformal VEM places degrees of freedom on boundary faces, we only have to duplicate those boundary faces. By contrast, conformal VEM would require duplicating vertices and edges, which can have arbitrarily high valences. We discuss our duplication process, which naturally complements Mandoline's data structures, in Section 6.5.

4.2 Constructing the Inner Product

Let us return to our discussion of VEM machinery. Combining (7) and (8) we have

$$f \approx \sum \langle \phi_i, f \rangle \, \phi_i = \sum m_i(f) \phi_i. \tag{9}$$

We will call values of $m_i(f)$ the $\mathcal{V}$-coefficients, since they are the coefficients for the virtual basis functions ϕ_i.

Although $\mathcal{V}$, with basis functions ϕ_i, is ostensibly our "primary" function space, it is ultimately dictated by the polynomial space $\mathcal{P}$: that is, while the placement of DOFs and domains c_i are determined by the mesh geometry, the choice of the g_i depends on $\mathcal{P}$ as we outline below. Together they define the m_i operator, and m_i in turn *implicitly* defines ϕ_i through the inner product of (8).

More specifically, the c_i and g_i will be defined so as to guarantee that the result of integrating by parts on $\int (\nabla \psi_\ell) \cdot (\nabla \psi_k)$, needed later for the Laplacian, can be represented in terms of $m_i(f)$. As we discuss in Section 5.3, this construction allows one to estimate

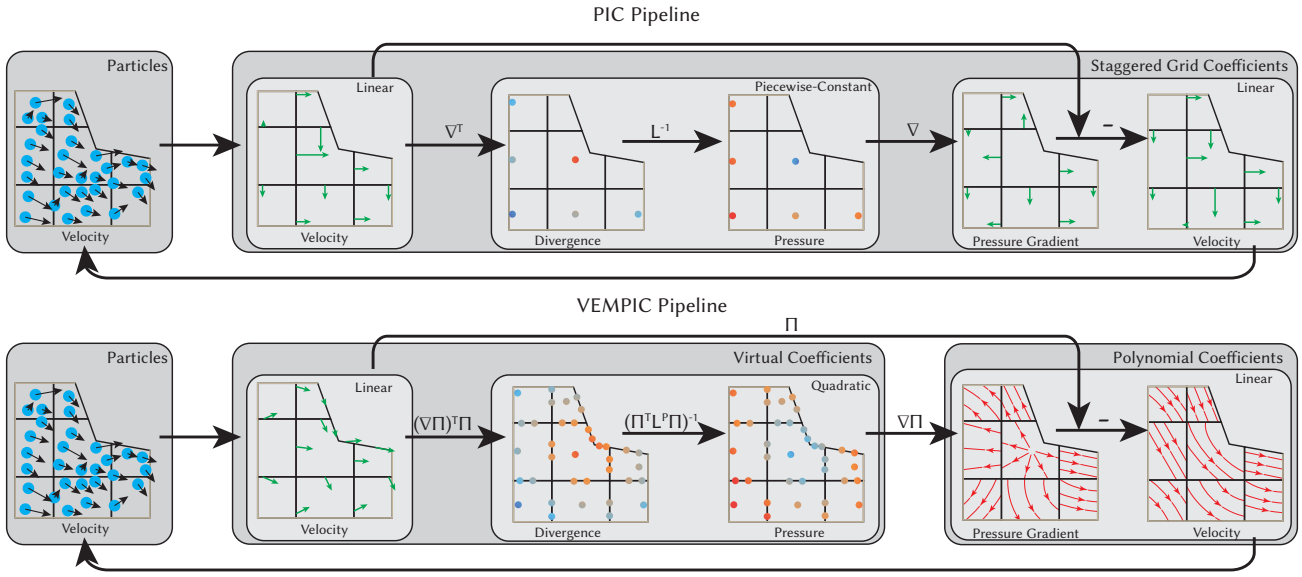


Fig. 2. The data pipeline of our simulator resembles that of a traditional PIC simulator. The main difference is that we store mesh data as either m_i-measured per-face $\mathcal{V}$-coefficients or Π-projected per-cell $\mathcal{P}$-coefficients. Our pressure system uses function spaces one degree higher than those used for the velocity.

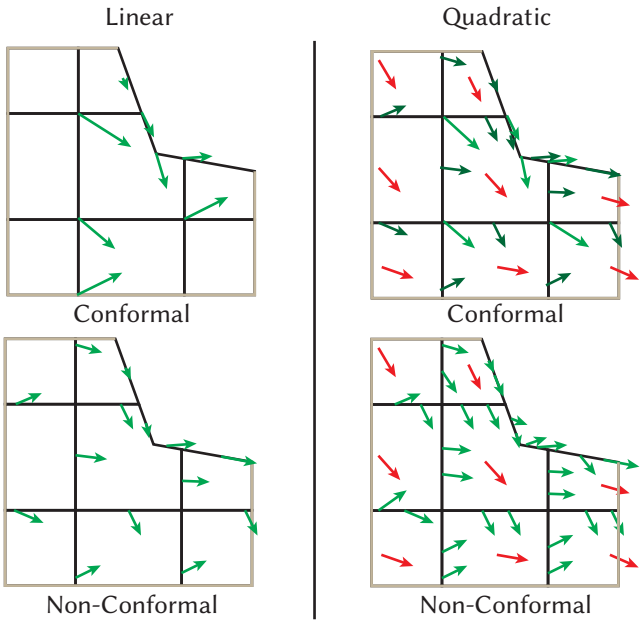


Fig. 3. In two dimensions, conformal VEM stores DOFs at mesh vertices in the linear case and adds extra DOFs on edges and cells for higher order variants. The non-conformal variant we use stores DOFs on edges in the linear case and similarly adds DOFs on edges and cells for higher order. The positions of arrows in the non-conformal case are not indicative of the velocity at a particular point, as those DOFs are integrals over faces. In three dimensions, boundary DOFs lie on faces of polyhedra, rather than edges.

$\int(\nabla\phi_i)\cdot(\nabla\phi_j)$, by first finding its $\mathcal{V}$-coefficients through evaluation of the m_i, and then performing the actual computation in terms of $\mathcal{P}$-coefficients.

Integration by parts transforms an integral over the domain Ω into a sum of integrals over the volume Ω and its boundary $\partial\Omega$:

$$\int_\Omega (\nabla\psi_\ell)\cdot(\nabla\psi_k)dV = \int_{\partial\Omega}\psi_\ell\nabla\psi_k\cdot dS - \int_\Omega \psi_\ell\Delta\psi_k dV. \qquad (10)$$

Therefore, for a given polyhedral cut-cell, there must be integration domains c_i for each of the cell's boundary faces and its interior. The g_i are simply polynomials chosen such that they span $\{\nabla\psi_j\}$ on $\partial\Omega$ and $\{\Delta\psi_j\}$ on Ω.

As an example in two dimensions, if we select $\mathcal{P}$ to be the space of quadratics then for a boundary edge e of a cell, we need two (virtual) basis functions ϕ_i defined over e to span the space of linear functions. To evaluate their associated m_i, we define two pairs c_n, g_n and c_o, g_o, where $c_n = c_o = e$ and $\text{span}(\{g_n, g_o\})$ is the space of linear functions on e. The interior space follows analogously.

4.3 Placement of DOFs

We use *linear* VEM for velocities and *quadratic* VEM for pressures, which means that for velocities the associated $\mathcal{P}$ is the space of linear functions and for pressures it is the space of quadratic functions.

Linear VEM requires only a single degree of freedom on each of the boundary faces of a cell (and none on the interior). Since the associated g_i on each face is the unit function (i.e., $g_i = 1$), (8) implies that, for a function f, the coefficient $m_i(f)$ must be the average of f on the face.

For quadratic VEM, each face has up to linear g_i and each cell has a single constant g_i (see Figure 4). Thus, in three dimensions there are three boundary m_i per face, and each evaluates one axis of the first moment of f with respect to a center point on the face.

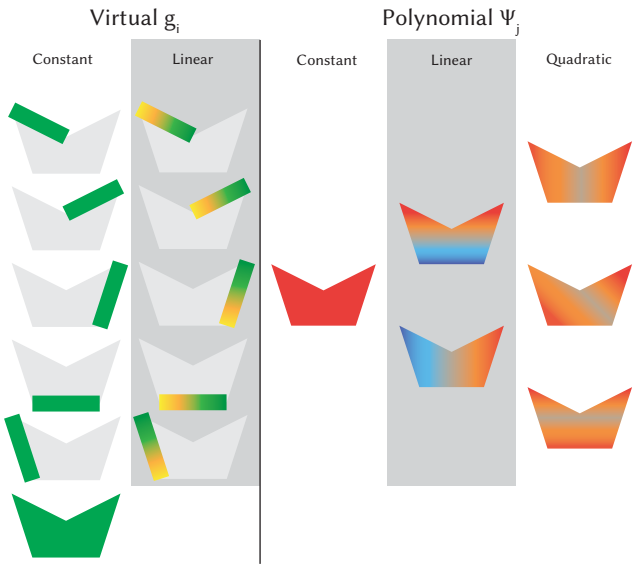


Fig. 4. When $\mathcal{P}$ is the space of quadratic polynomials, the g_i functions span linear polynomials along each boundary facet and have a constant function on cell interiors. These g_i implicitly define $\mathcal{V}$ via Equation (8).

For example, the integrands of (8) for the x-moment DOFs have the form

$$g_i(x,y,z)f(x,y,z) = (x - C_x)f(x,y,z), \tag{11}$$

where C_x is the x-coordinate of the center. The single per-cell interior DOF is the mean value over the cell. We provide explicit definitions for the g_i polynomials in Section 6.4.

4.4 The Projection Operator

We stated that the VEM projection operator Π projects a function represented in the virtual space $\mathcal{V}$ into one represented in the explicit polynomial space $\mathcal{P}$. This means that given the $\mathcal{V}$-coefficients m_i for a given input function f with respect to a $\mathcal{V}$-basis ϕ_i, we can approximate it in terms of the $\mathcal{P}$-basis functions, ψ_j using:

$$f \approx \sum_i m_i(f)\phi_i \approx \sum_i \sum_j (\Pi_{ij} m_i(f))\psi_j. \tag{12}$$

Thus the $\Pi_{ij} m_i(f)$ are the coefficients in the $\mathcal{P}$-basis. The polynomials ψ_j are known explicitly (§5.2), so given $m_i(f)$ and Π this approximation of f can be directly evaluated at any point. We detail the construction of this projection in Section 5.3.

VEM's virtual basis functions and projection operator, outlined above, lay the foundation for our hybrid VEMPIC scheme detailed in the following section. Specifically, representing the velocity field as in (12) allows it to be naturally point sampled and integrated for advection and transfer operators, and will enable us to define the discrete divergence and gradient operators required for our novel VEM-based pressure projection.

5 VEM PIC

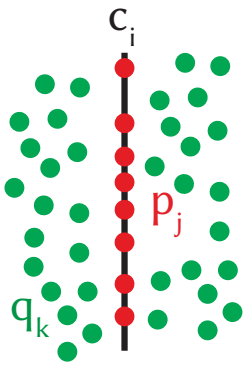
We now describe how to leverage VEM and the cut-cell mesh within our particle-in-cell framework. At the outset of each timestep, we have a collection of particles, which act as point samples of the smooth vector field representing the fluid velocity. We migrate this velocity data onto the mesh, perform mesh-based pressure projection, and finally transfer the updated velocity field back to the particles. We begin with the particle transfer operations before considering the VEM-based pressure projection.

5.1 Particle-to-Mesh Transfer

Transferring data from particles to the mesh entails finding $\mathcal{V}$-coefficients for the mesh's (virtual) basis functions based on particle values. Although the $\mathcal{V}$ basis functions cannot be evaluated, their coefficients can be determined by discretizing the m_i via numerical integration. Equation (8) shows that for an arbitrary function f, $m_i(f)$ is the mean of the product fg_i within the domain c_i. This results in a relatively straightforward rule for approximating m_i from a set of sample points $\{p_j\} \in c_i$:

$$m_i(f) = \frac{1}{|c_i|}\int_{c_i} g_i f \approx \frac{1}{N}\sum_j g_i(p_j) f_j, \tag{13}$$

where the $f_j = f(p_j)$ are the known data stored on the particles. This expression is simply the mean value of $g_i f_j$ for each c_i, which can be straightforwardly evaluated (recalling that the g_i are known polynomials).



In general, there exist m_i for cells and cell boundaries, and the sample points used in each case must be handled differently based on the availability of samples. When c_i is a cell, the fluid particles already lie within it so we could directly use them as point samples to evaluate (13); however, linear VEM , which we use for velocity, has no interior DOFs so we do not need to.

When c_i is a boundary face we need to evaluate a boundary integral, but particles almost never lie precisely on the boundary. Instead, we must create sample points whose values are estimated from the nearby particle data. We use radial basis interpolation on the fluid particles, which has previously been used for particle-to-grid transfers [Batty and Bridson 2008; Ando et al. 2013]. An input signal f is approximated by

$$\tilde{f}(p) = \sum_i \frac{f_i W(\|p - q_i\|)}{\sum_k W(\|p - q_k\|)}, \tag{14}$$

where W is a radial basis function, p is a point that we want to query, q_i are the particle locations, and f_i are known values of f at those particle positions. We can then estimate the boundary coefficients by evaluating $m_i(\tilde{f})$, using sample point locations drawn from a uniform distribution on the surface of each boundary face. The radial basis functions are used only for this transfer and play no other role in our simulator.

Evaluating boundary samples from radial basis interpolants in this way generalizes the standard PIC-style particle-to-grid transfer; there an equation like (14) is applied to the center of each face, which is equivalent to using a single sample in our scheme or a

0-th order quadrature rule. For the radial basis function we chose the cubic-spline of Desbrun and Gascuel [1996] with a radius of one-tenth the grid edge length to strike a balance between sub-grid detail and performance.

5.2 Mesh-to-Particle Transfer

The polynomial basis $\mathcal{P}$ is used to evaluate the velocity field at arbitrary points in the simulation domain, both for advection and when transferring the mesh velocities onto particles. Intuitively, $\mathcal{P}$ provides local Taylor series expansions of the velocity field, while the cell geometry determines the spacing and domains of these local approximations. Concretely, for each cell we let $\mathcal{P}$ be the space of polynomials up to degree k, and choose its basis $\{\psi_j\}$ to be the space of *scaled* monomials up to degree k. Per de Dios et al. [2016], we therefore define our polynomial basis as the product

$$\psi_\alpha(p) = \prod_{j=1}^{3} \left(\frac{(p - C_i)_j}{D_i} \right)^{\alpha_j}, \tag{15}$$

where p is a point in space, j loops over coordinate axes, $\alpha \in \mathbb{N}^3, |\alpha| \leq k$ are the polynomial exponents, C_i is the center of the cell c_i, and $D_i = \max_{p \in c_i} (p - C_i)$ is the diameter of the cell. Given the $\mathcal{P}$-coefficients for our velocity field we can now evaluate it at any point in a cell.

The remaining missing piece is the VEM projection operator Π from $\mathcal{V}$ to $\mathcal{P}$. It will fall out naturally from defining our pressure projection in the next section. We will then be able to convert a particle-based field f to the corresponding explicit polynomial representation in accordance with (12), as follows. Construct an estimate of f, denoted $\tilde{f}$, from the discrete particle data $\{f_\ell\}$ using our radial basis functions. Then, evaluate $m_i(\tilde{f})$ to convert this estimate to $\mathcal{V}$-coefficients, and finally apply Π to project them to $\mathcal{P}$-coefficients.

5.3 Pressure Projection

The VEM Laplacian $\mathbf{L}$ is a sparse matrix that *estimates* the $\mathcal{V}$-Laplacian $\mathbf{L}^{\mathcal{V}}$:

$$\mathbf{L}_{ij} \approx \mathbf{L}^{\mathcal{V}}_{ij} = \int (\nabla \phi_i) \cdot (\nabla \phi_j). \tag{16}$$

Since the ϕ_i are virtual, we instead construct it by projecting the $\mathcal{V}$-coefficients to corresponding $\mathcal{P}$-coefficients and evaluating an analytically computed Laplacian $\mathbf{L}^{\mathcal{P}}$ over $\mathcal{P}$. Thus, for arbitrary functions $s, t \in \mathcal{V}$, we define $\mathbf{L}$ in terms of the $\mathcal{P}$-Laplacian $\mathbf{L}^{\mathcal{P}}$ and the (yet to be defined) projection Π by

$$s^T \mathbf{L} t = (\Pi s)^T \mathbf{L}^{\mathcal{P}} \Pi t. \tag{17}$$

The $\mathcal{P}$-Laplacian can be written coefficient-wise as

$$\mathbf{L}^{\mathcal{P}}_{ij} = \int (\nabla \psi_i) \cdot (\nabla \psi_j) \tag{18}$$

and in matrix form as

$$\mathbf{L}^{\mathcal{P}} = \left[\nabla\right]^T \begin{bmatrix} \mathbf{M} & & \\ & \mathbf{M} & \\ & & \mathbf{M} \end{bmatrix} \left[\nabla\right] \tag{19}$$

where $\mathbf{M}$ is the mass matrix with entries $\mathbf{M}_{ij} = \int \psi_i \psi_j$ and $\left[\nabla\right]$ is a discrete gradient operator whose rows are the partial derivatives of ψ_j in terms of other ψ_k coefficients. For brevity, going forward we will let $diag(\mathbf{M}, 3)$ denote the block-diagonal matrix composed of three copies of $\mathbf{M}$.

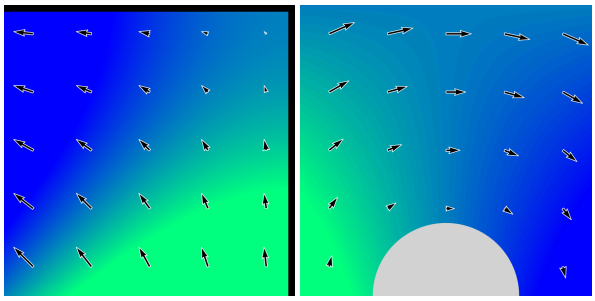
We make use of the construction above during pressure projection, i.e., when determining pressure as a minimizer of (4). The necessary condition can be written as inner products over functions via

$$\langle \nabla \lambda, \nabla p \rangle = \langle \nabla \lambda, u \rangle \quad \forall \lambda \in \mathcal{P}. \tag{20}$$

To fully discretize the above equation using the $\mathcal{V}$-coefficients available to us, we must map everything to polynomials by applying the projection operator Π. Doing so transforms the necessary condition into a linear system,

$$\Pi^T \mathbf{L}^{\mathcal{P}} \Pi p = \Pi^T \left[\nabla\right]^T \operatorname{diag}(\mathbf{M}\Pi, 3)\mathbf{u}, \tag{21}$$

where $\mathbf{u}$ is a vector comprising three sets of VEM-coefficients stacked upon one another, one for each coordinate axis. In this equation, the left hand side has two derivatives applied to it whereas the right hand side only has one.



(a) Corner Cell (b) Cut-Cell

Fig. 5. Quadratic pressures allow us to better resolve boundary conditions. Up to a choice of origin, $\nabla((x+y)(x-y))$ flows around corners (left) and $\nabla(xy)$ flows around obstacles (right).

5.3.1 Quadratic Pressure. To ensure that the polynomial space used for the pressure gradient is of the same order as the polynomial space used for u, the space for p must be one order higher. Since we have chosen a linear velocity space, we therefore use a quadratic VEM formulation for our pressure projection step, consistent with Section 4.3. Ando et al. [2013] similarly used piecewise-linear pressures and piecewise-constant velocities, which they noted reduces locking artifacts. This choice also allows our pressure solver to better resolve flows around corners (Figure 5a) and non-trivial boundaries (Figure 5b).

5.3.2 Projection Operator. Let us finally define the VEM projection operator Π that projects a function $f \in \mathcal{V}$ into the span of $\mathcal{P}$. If $f \in \mathcal{V}$ and $\Pi f \in \mathcal{P}$, then there exist coefficients $f_i, \tilde{f}_j$ such that $f = \sum f_i \phi_i \in \mathcal{V}$ and $\Pi f = \sum \tilde{f}_j \psi_j \in \mathcal{P}$. This operator is defined independently for each cell, so for this section the operators will refer to a single cell. VEM determines the $\tilde{f}_j$ coefficients according

to the following minimization with respect to Π:

$$\int_c \|\nabla(f-\Pi f)\|^2 = \int_c \sum (\nabla\phi_i)\cdot(\nabla\phi_j) f_i f_j - 2\sum (\nabla\phi_i)\cdot(\nabla\psi_j) f_i \bar{f}_j + \sum (\nabla\psi_i)\cdot(\nabla\psi_j)\bar{f}_i\bar{f}_j. \tag{22}$$

That is, Π maps functions in $\mathcal{V}$ to the closest function in $\mathcal{P}$ using the Dirichlet semi-norm to measure distance. The use of the Dirichlet semi-norm (which emphasizes gradients rather than values of f) reflects the fact that following the VEM solve for pressure, we will subtract a $\mathcal{P}$-pressure gradient from a $\mathcal{P}$-velocity field to achieve incompressibility. Thus, this choice of projector allows us to optimize for the $\mathcal{P}$-pressure gradient directly while using the $\mathcal{V}$ degrees of freedom with a certain degree of continuity between cells. For a single cell, projecting the minimal semi-norm $\mathcal{V}$ function to $\mathcal{P}$ results in the minimal semi-norm function in $\mathcal{P}$. However, for multiple cells, neighboring cells share $\mathcal{V}$ coefficients so the result of projecting a $\mathcal{V}$-minimal solution implicitly balances $\mathcal{P}$-minimality with the magnitude of the discontinuity between cells [Brezzi and Marini 2014].

If we momentarily ignore the kernel of $L^{\mathcal{P}}$, the necessary conditions of (22) imply that for any $\lambda = \sum \lambda_i\psi_i \in \mathcal{P}$,

$$0 = \int_c \sum_k \lambda_k(\nabla\psi_k)\cdot\left(\sum_i f_i\nabla\phi_i - \sum_j \bar{f}_j\nabla\psi_j\right). \tag{23}$$

Recalling (18), we can express the above equation in matrix form as

$$0 = \lambda^T\left(\mathbf{G}f - \mathbf{L}^{\mathcal{P}}\bar{f}\right). \tag{24}$$

Since λ is arbitrary, we need $\mathbf{G}f - \mathbf{L}^{\mathcal{P}}\bar{f} = 0$. Thus we would like to define the projection operator as

$$\Pi = \left(\mathbf{L}^{\mathcal{P}}\right)^{-1}\mathbf{G}, \tag{25}$$

but there are two tasks required before we can achieve this: we need to define $\mathbf{G}$ and modify $\mathbf{L}^{\mathcal{P}}$ to be invertible, yielding a new matrix $\hat{\mathbf{L}}^{\mathcal{P}}$.

The virtual basis functions ϕ_i are implicitly defined to enable a particular definition of $\mathbf{G}$ that is computable and relates the ψ_j with m_i. The coefficients of G are defined so that for $f \in \mathcal{P}$, the value of $\langle\nabla\psi_i, f\rangle$ can be computed in terms of $\mathcal{V}$-coefficients $m_k(f)$. Equation (9) implies $\mathbf{G}$ can be defined such that

$$\left\langle\nabla\psi_i, \nabla\psi_j\right\rangle = \left\langle\nabla\psi_i, \sum_\ell m_\ell(\nabla\psi_j)\phi_\ell\right\rangle = \sum_k \mathbf{G}_{ik}m_k(\psi_j). \tag{26}$$

The above equality defines requirements for the g_i used to define the virtual basis functions. In particular, g_k on boundary faces must satisfy

$$N\cdot\nabla\psi_j \in Span(\{g_k\}), \tag{27}$$

where N is the normal of a boundary face, and interior per-cell g_ℓ must satisfy

$$\Delta\psi_j \in Span(\{g_\ell\}). \tag{28}$$

Therefore when $\mathcal{P}$ spans degree d polynomials the boundary g_k must span at least $d-1$ degree polynomials over their boundary faces and the cell g_ℓ must span at least $d-2$ degree polynomials over their interiors. Non-conformal VEM defines its per-face g_k as monomials on boundaries (see Section 6.4) and g_ℓ in cells as the subset of monomials ψ_j that spans the space of $d-2$ polynomials. The algebraic manipulations that convert ψ_j to g_k and g_ℓ determine the coefficients of $\mathbf{G}$ are detailed in Appendix A. The result is

$$\left\langle\nabla\psi_i, \nabla\psi_j\right\rangle = \sum_k \xi^{\nabla}_{k,i}|c^k|m_k(\psi_j) - \sum_\ell \xi_{\ell,i}|c^\ell|m_l(\psi_j), \tag{29}$$

where the ξ^{∇} and ξ arise from the expansions

$$N\cdot\nabla\psi_i = \sum_k \xi^{\nabla}_{k,i}\psi_k, \tag{30}$$

$$\Delta\psi_i = \sum_\ell \xi_{\ell,i}\psi_\ell \tag{31}$$

Equation (30) is defined along boundary faces and Equation (31) is defined in each cell. Since these expressions are geometry-dependent they do not have a simple, general closed-form, but can nevertheless be straightforwardly evaluated for each particular cell geometry. Our reference implementation demonstrates the details of this process.

5.3.3 Removing the kernel. Before we can compute the projection operator for (25) we must make the Laplacian $\mathbf{L}^{\mathcal{P}}$ invertible. The root of this issue is that the first vector of the $\mathcal{P}$ basis, ψ_0, is a constant function and $\langle\nabla\psi_0, \nabla p\rangle = 0$ for all p, which causes the first row and column of $\mathbf{L}^{\mathcal{P}}$ to be entirely composed of zeros. In fact, the first row of $\mathbf{G}$ is also zero for the same reason, so both matrices need to be modified to guarantee that Π can project constant functions to constant functions. This is done by adding terms to the first row of both $\mathbf{L}^{\mathcal{P}}$ and G so that the first entries of $\mathbf{L}^{\mathcal{P}}f$ and $\mathbf{G}f$ evaluate to an average-like statistic that is consistent with (26). In particular, we let the first entry of the vector $\mathbf{G}[m_i(f)]$ be the average value of f in the cell and let the first row of our modified Laplacian be the average values of each ψ_j in each cell.

The type of average computed depends on the VEM degree. For quadratic VEM, there is a degree of freedom indexed by k that is defined as the average value (i.e., $m_k(f) = {}^1/_{|c|}\int_c f$ for cell c and arbitrary function f), so the first row of $\mathbf{G}$ only needs a single 1 on the kth column. However, for linear VEM no such degree of freedom exists so the average value used is the mean value along the boundary, ${}^1/_{|\partial c|}\int_{\partial c} f$. The modified $\mathbf{G}$ for linear VEM therefore has a first row of the form

$$\hat{\mathbf{G}}_{0,i} = \frac{|c_i|}{|\partial\Omega|}. \tag{32}$$

In order to satisfy Equation (26) the first row of the modified Laplacian $\hat{\mathbf{L}}^{\mathcal{P}}$ is defined as the average values for each basis function ψ_j.

5.3.4 Solving The Pressure Projection System. The Laplacian form we have described corresponds only to the *consistency term* in standard VEM [Beirão da Veiga et al. 2014]. VEM usually includes an additional regularizer (or *stability term*), which aims to minimize the discrepancy between $\mathcal{V}$-coefficients and the projected polynomial's measurements (i.e., $m_i(f) \approx m_i(\Pi f)$). This regularizer is typically necessary because $|\mathcal{V}| > |\mathcal{P}|$, causing the VEM Laplacian to have a rather large null space of $\mathcal{V}/\mathcal{P}$ (i.e., beyond the usual constant

function kernel). While many linear solvers cannot handle such problems, the conjugate gradient (CG) method is robust to such a null space, provided that our right hand side lies in the range of $\mathbf{L}^{\mathcal{V}}$. This is because, for semidefinite operators, CG only selects descent directions in the range of the operator [Hayami 2018]. Our right hand side lies in this range so CG can find a solution to our pressure system. One can see that the right hand side lies in this span by noting that

$$\min_p \mathcal{E}(p) = \min_p \|[\nabla]p - \text{diag}(\Pi, 3)u\|^2_{\text{diag}(M,3)}, \tag{33}$$

is a convex quadratic equation for which there exists p such that $0 = \frac{\partial \mathcal{E}}{\partial p}$, a necessary condition for a solution. Equation (21) is precisely the aforementioned necessary condition, so the system must be solvable by CG.

Because our system can be successfully solved without the additional stability term we simply neglect it. In fact, we found that CG failed to converge after two or three simulation steps with the stability term activated. We believe this is because the stabilizer, which has the form $(I-\Pi)^T(I-\Pi)$, will penalize regions where quadratic pressure fields cannot fully resolve valid flows like complex, potentially discontinuous boundaries.

5.4 Updating Velocity

The velocity field data computed from the pressure projection is only used to perform the advection phase (including updating particle data); it is not needed in the next frame. Since advection uses only the velocity field expressed in the explicit polynomial space $\mathcal{P}$, we do not need to recover its updated representation in $\mathcal{V}$. We therefore conclude our pressure projection by directly computing

$$\tilde{u}_{\mathcal{P}} \leftarrow \Pi u_{\mathcal{V}} - [\nabla]\Pi p. \tag{34}$$

That is, we find (only) the $\mathcal{P}$ coefficients for the new velocity field using a Π-projected velocity update.

At this point, we have described how we use VEM to create a PIC-based fluid simulator with complex geometric cut-cells. Building on top of the standard non-conformal VEM Laplacian of de Dios et al. [2016], we have formulated a consistent cut-cell VEM pressure projection, including an appropriate discrete divergence operator for the system's right-hand-side, and proposed a natural method for connecting particle representations of velocity fields with a non-conformal VEM representation. It remains to discuss how we leverage the structure of our cut-cells to create an efficient simulator.

6 GEOMETRIC QUERIES USING CUT-CELLS

The practical application of explicit polyhedral cut-cell meshes in simulation remains an under-explored topic and therefore several fundamental geometric queries required for our fluid solver have not been described in the literature. Fortunately, the underlying structure of cut-cell meshes often allows existing algorithms and data structures for triangle meshes and/or Cartesian grids to be used as accelerators, before applying a slower algorithm on a smaller localized set of cut-cells. This is analogous to separate chaining in hash tables, where the hash function determines a bucket of elements in constant time, and then the entries of the bucket are traversed linearly. We consider several such queries below.

6.1 Cut-Cell Mesh Representation

Although the cut-cells in a cut-cell mesh are arbitrary polygons, their boundary polygons, edges, and vertices, which we call cut-faces, cut-edges, and cut-vertices, respectively, are all associated with (and contained within) elements of the Cartesian grid planes and original surface triangles used in their construction. It is these relationships that we employ for efficient lookups.

6.1.1 Cut-cells. Consistent with the *Mandoline* library [Tao et al. 2019], the *topology* (connectivity) of cut-cells is expressed using a matrix that maps cut-cell indices to associated cut-face indices (with signs to indicate orientation). The actual *geometry* of the cut-cells is determined by the cut-faces that make them up. The algorithms used to query or manipulate these cut-faces can be specialized according to their classification into one of two types: triangle cut-faces, which are convex polygons that lie within one of the input surface triangles, and grid cut-faces, which are general planar polygons that lie within one of the axis-aligned cutting planes. For instance, triangle fans can easily triangulate cut-faces that emerge from triangles because such cut-faces are convex, whereas grid cut-faces may be non-convex and even have holes, and thus require more complex algorithms.

6.1.2 Cut-faces. All cut-faces are planar polygons composed of one outer boundary loop and potentially several interior boundary loops for holes. Each polygon consists of a list of boundary loops, and each boundary loop is an ordered list of indices into the set of cut-vertices. Triangle cut-faces additionally store a reference to the triangle that generated them and the associated barycentric coordinates for their vertices. These coordinates provide a useful parameterization for point positions within the plane of the triangle cut-face; for a grid cut-face the 2D coordinates of its containing axis-aligned plane play the same role.

6.1.3 Cut-edges. Cut-edges consist of a pair of cut-vertices (i.e., its two endpoints) and the positions of those cut-vertices within the grid edge or triangle edge that created it (its *parent*). Specifically, the position of a point p is represented by the value t, such that $(1-t)v_i + tv_j = p$, where the parent edge has endpoints v_i, v_j. Similar to the barycentric coordinates for triangle cut-faces, these positions are useful for performing computations along the parent edges.

6.2 Locating Elements in Cut-Cell Meshes

Identifying the element containing a point is an important task for particle-in-cell algorithms. For uniform Cartesian grids, this point-location problem is $O(1)$ due to the existence of an efficient hashing function (e.g., the coefficient-wise floor function); for triangle meshes bounding volume hierarchies can provide logarithmic searches for the nearest triangles, which, with a sign convention, can be used to determine whether a point lies in a given cut-cell. As a concrete example, determining the nearest cut-edge to a point can be implemented by first finding the nearest parent triangle edge or grid edge using an off-the-shelf implementation, and then finding the nearest cut-edge by iterating through the cut-edges that the parent contains (Figure 6).

6.2.1 Integration. We use integral-based techniques (i.e., winding numbers) to detect if a point lies in a cut-cell because, as detailed

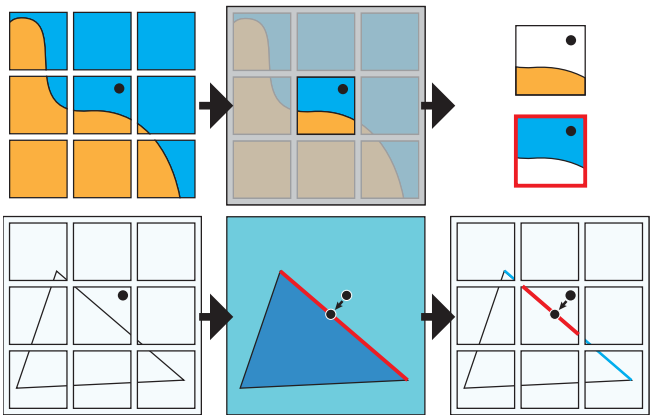


Fig. 6. Top: To identify which cut-cell a point (black disk) lies in, we first identify which regular grid-cell it lies in and then iterate over all cut-cells within that grid-cell, using generalized winding numbers to test if the point is inside. Bottom: To determine the nearest boundary point to a query point (black disk), we first find the nearest input parent edge before iterating over the cut-edges it contains to determine the nearest one.

below, such techniques typically do not require sophisticated algorithms for triangulation or tetrahedralization of the geometry. Properly triangulating our cut-cells would be challenging due to potential non-convexity or holes. Instead, when evaluating two-dimensional integrals on polygonal boundaries of cut-cells, we triangulate the boundaries with triangle fans; for three-dimensional integrals over polyhedral cut-cells, we extend the boundary triangles to tetrahedra by connecting each triangle to the cell centroid. Although such a meshing will often possess intersections and inversions, linearity of integration implies that the integrals still yield correct results, as was also exploited by Edwards and Bridson [2014].

6.3 Advection

A key opportunity to exploit the parent geometry of our cut-mesh is in the advection of particles, since it involves evaluating the velocity field at arbitrary points (which requires knowing their containing cut-cell) and handling potential collisions with obstacles. Both computations can be accelerated using additional two-step processes that again leverage the original cut-cell-generating geometry.

In a typical simulator, advection requires identifying the Cartesian cell containing a particle and applying grid-based interpolation to sample the velocity; collision-handling then uses an obstacle's interpolated signed distance field to project out penetrating particles. We will describe how our cut-cell approach trades additional computational effort in these steps for higher fidelity fluid boundary behavior that surpasses the capabilities of basic grids.

We employ a second order Runge-Kutta integrator to advect particles. We use Embree's raytracing kernels [Wald et al. 2014] to detect when a particle's proposed path intersects with obstacles and for simplicity apply mirror reflection to bring them back into the fluid domain. The two types of obstacles that arise are the outer grid boundary and the triangle mesh(es), so we aggregate them into a single triangle mesh to pass into Embree. Since this query uses only obstacle information, the cut-cell mesh is not needed.

Table 2. The polynomial degree of each type of degree of freedom, with the number of degrees of freedom required per element in parentheses. For instance, in each cell the pressure $\mathcal{P}$ was of degree 2 and required 10 degrees of freedom per cell. There are no $\mathcal{V}$-cell velocity degrees of freedom.

Type	$\mathcal{P}$-Cell	$\mathcal{V}$-Boundary	$\mathcal{V}$-Cell
Velocity Degree(DOFs)	1(4)	0(1)	
Pressure Degree(DOFs)	2(10)	1(3)	0(1)

6.3.1 Nearest-Cell Queries. Mirror reflections can induce multiple bounces near corners, which can be costly to resolve due to the multiple raycast operations required. Rather than let individual particles induce multiple raycasts, if a collision is detected we perform a single mirror reflection and check whether the reflection caused the particle to escape the valid domain. If so, we project the escaped particle back into the simulation domain via a nearest-cell query.

Nearest-cell queries proceed by finding the closest grid- and triangle-elements (vertices, edges, faces, and cubes), looping through the cut-mesh elements that they contain, looking up the associated cut-cells that use those elements, and selecting the cut-cell that is closest among them to the query point (Figure 6). (For grid-elements we directly compute the nearest elements with modulo arithmetic and for triangle-elements we use an axis-aligned bounding box hierarchy.)

For the last step above, we determine the closest cut-cell from the set of candidate cut-cells by finding the one with the closest cut-face. Finding the nearest cut-face can be performed as follows, without needing a triangulation of the faces. For triangle cut-faces, we find the nearest triangle, project the query point onto its surface, and then determine the containing grid cell of the projected point. Since the intersection of a triangle and cube yields a single polygon (by convexity), this information identifies the unique polygonal cut-face. For grid cut-faces, we project the query point onto the closest axis-aligned plane of the grid, and use the winding number within the plane to determine the cut-face containing the point.

6.4 Bases and Degrees of Freedom

We have four basic types of coefficients because both pressure and velocity will require their own $\mathcal{V}$ and $\mathcal{P}$ spaces. Both $\mathcal{V}$ and $\mathcal{P}$ are composed of per-cell coefficients and $\mathcal{V}$ has per-boundary coefficients as well, making a total of five types of degrees of freedom, as shown in Table 2. Below we discuss some subtleties of the polynomial spaces, with ψ_j being the polynomials used for $\mathcal{P}$ and g_i the polynomials for $\mathcal{V}$. (Recall that although $\mathcal{V}$ is "virtual", the definition of its coefficients requires the g_i functions, as in Section 4.2).

For each cut-face we construct a basis of scaled monomials, similar to each cell's $\mathcal{P}$ in (15). For exponents α, β we define $g_{\alpha,\beta}(u,v) = \frac{u^\alpha v^\beta}{D^{\alpha+\beta}}$, where D is the diameter of the cut-face. This in-plane definition of $g_{\alpha,\beta}$ is mapped to three dimensions with the assistance of a local frame $F : \mathbb{R}^2 \to \mathbb{R}^3$ defined by

$$F(u,v) = Xu + Yv + C, \tag{35}$$

for vectors $X, Y, C \in \mathbb{R}^3$ where X, Y are some orthogonal basis and C is the center of a face. For cut-faces derived from grid faces on axis d_i

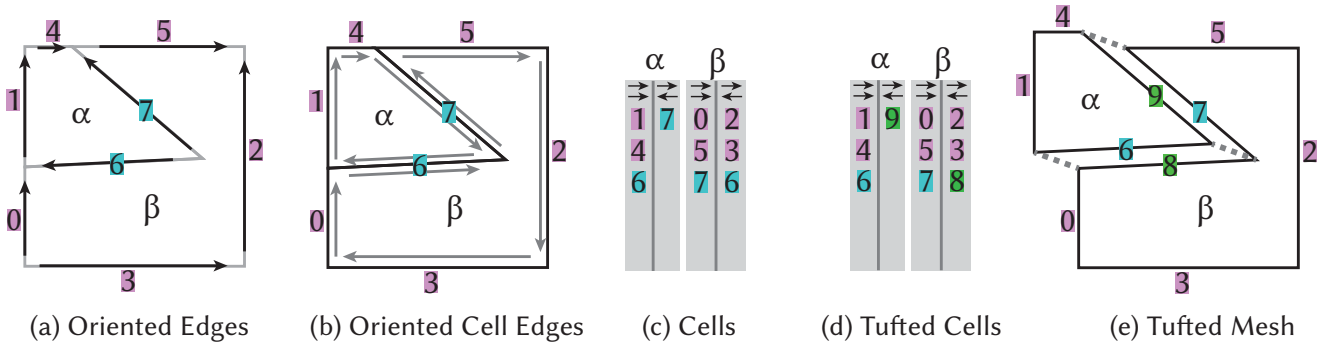


Fig. 7. In two dimensions, each edge of our cut-cell mesh has its own orientation (a) and each cell has an orientation for its boundary faces (b), indicated by black and gray arrows, respectively. Mandoline encodes each cell using the indices of its boundary edges and the relative orientations of those boundary edges (c); whether these two orientations agree or not is visually depicted by the two columns for each cell. We create a tufted mesh by duplicating/relabeling every input geometry edge (teal) that has an opposing orientation (d). These new edges (green) topologically separate the cells according to the input geometry (e). This process generalizes to three dimensions using surface normals rather than edge directions.

we use unit vectors $\vec{e}_{(i+1)\%3}$ and $\vec{e}_{(i+2)\%3}$ as our frame and for those coming from triangles we arbitrarily choose the orthogonalization of the first two edges.

The evaluation of $\xi^{\nabla,d}_{i,j}$ in Equation (29) requires a change of basis from ψ_i to g_k, which is computed by writing ψ_i in terms of boundary frames:

$$\psi_{\alpha,\beta,\gamma}(F(u,v)) = \sum_{\bar{\alpha}+\bar{\beta}\leq\alpha+\beta+\gamma} \xi_{\alpha,\beta,\gamma;\bar{\alpha},\bar{\beta}} \frac{u^{\bar{\alpha}} v^{\bar{\beta}}}{D^{\bar{\alpha}+\bar{\beta}}}, \tag{36}$$

where α, β, γ are the exponents of the monomial in $\psi_{\alpha,\beta,\gamma}$. We computed these coefficients by algebraically expanding $\psi_{\alpha,\beta,\gamma}(F)$ to coefficients of $u^{\alpha}v^{\beta}$ monomials and multiplying those coefficients by $D^{\bar{\alpha}+\bar{\beta}}$.

6.5 Two-sided Flows and Non-manifold Geometry

For each obstacle face we maintain two degrees of freedom, similar to the tufting process used by Sharp and Crane [2020], which allows for independent fluid flows on either side of a mesh. Because our degrees of freedom are stored on faces, this immediately allows us to also simulate on meshes with non-manifold vertices or edges.

Rather than trying to detect which cut-faces lay on non-manifold boundaries (and thus need duplication), we simply create two sets of DOFs for every cut-face originating from the input triangle mesh. The Mandoline boundary representation (Figure 7) makes it straightforward to identify which set of degrees of freedom a cut-cell should use: we simply rewrite the indices associated with cut-faces whose orientation differs from that of the underlying mesh.

7 RESULTS

We tested our VEMPIC simulator with a variety of complex meshes in both two and three dimensions. In this section we discuss the types of challenging geometric features that our method successfully resolves. All simulations were run on a AMD Ryzen 9 5900X with 64GB of RAM using Mandoline [Tao et al. 2019] to generate the meshes, Eigen [Guennebaud et al. 2010] for linear algebra, OneTBB [one 2020] for parallelization, and Embree [Wald et al. 2014] to detect particle-boundary collisions. An additional attractive feature of our method is that it did not require implementing cell-merging, as used by other cut-cell simulators [Edwards and Bridson 2014; Azevedo et al. 2016]. To aid reproducibility, our reference implementation can be found at https://github.com/mtao/vempic.

7.1 Scenes

Our simulation scenarios were all created by placing a piece of test geometry in a cubic domain with a smoke emitter at the bottom, e.g., as in Figure 8. The resulting simulation domain is instantiated with particles distributed uniformly using Bridson's Poisson disk sampler [Bridson 2007]. The emitter region converts any particles that enter its domain into smoke particles, which are buoyant and therefore rise. We typically used a 60^2 grid in 2D and 50^3 in 3D; despite these modest resolutions our method nevertheless resolves the details of various complex obstacles. Table 3 lists data, timings, and some features for all of our scenes.

We provide two visualizations of each scene: a traditional smoke rendering and a velocity visualization using the Winter colormap. In two dimensions we draw only smoke particles to produce the smoke rendering, and draw both smoke and air particles for our velocity visualization. In three dimensions we render smoke plumes using Blender [Blender Online Community 2018]. Particles used for the velocity visualization are selected in various ways to highlight different features of our simulation, such as particles near the occluding geometry or the fastest particles. We generated streamlines by selecting a subset of particles and tracing their trajectories over multiple timesteps. The particles chosen are the fastest moving particles in some intermediate frame and in different figures we filter according to locations of interest, such as the surface of a bunny (Figure 8), a dragon's mouth (Figure 11), or the interior of a monkey's skull (Figure 14). In some examples we only filter for fast particles, which occasionally resulted in streamlines further from the obstacle (Figure 15). Because we also randomly remove a

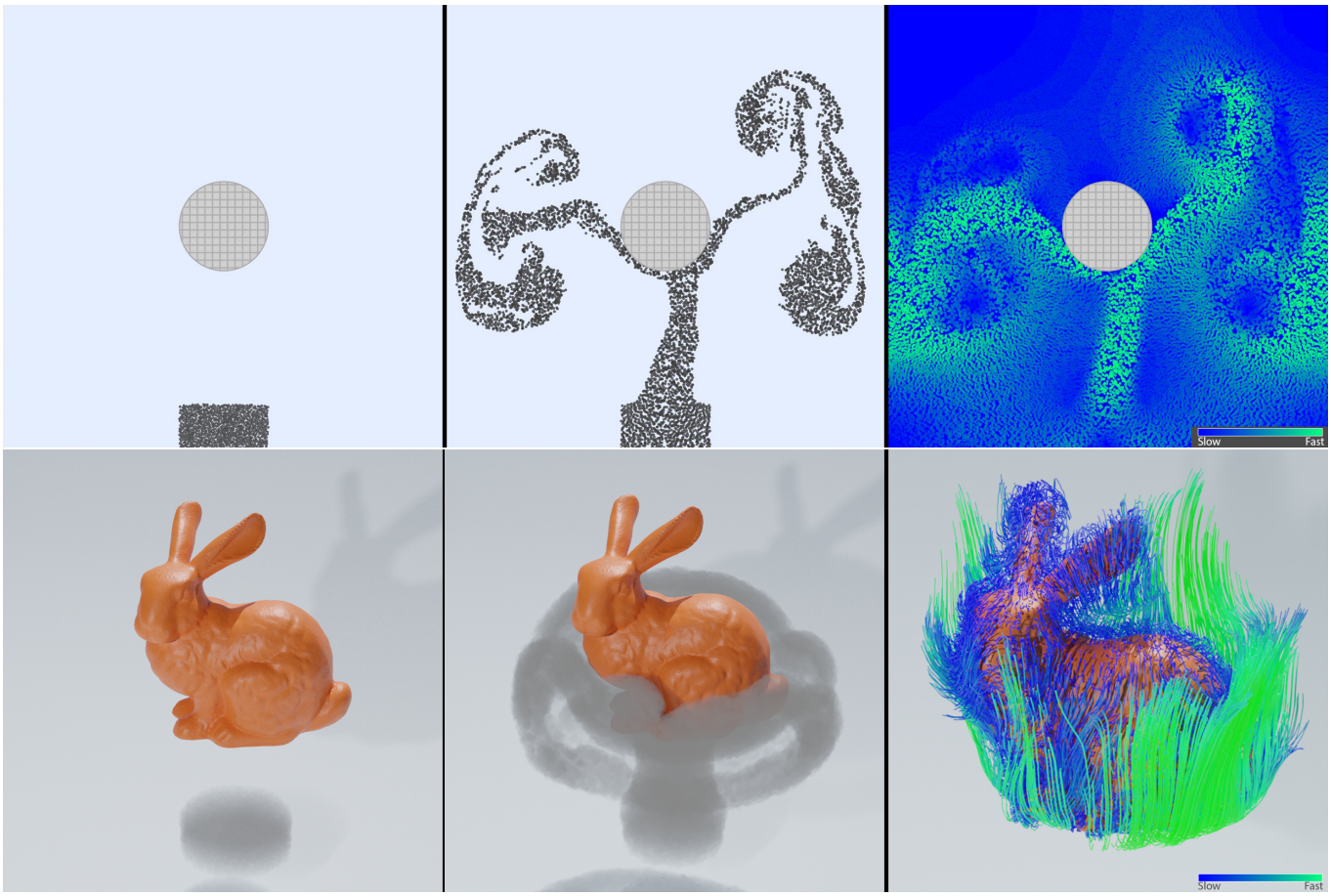


Fig. 8. Smoke is emitted from a rectangle (resp. cylinder) at the bottom of the domain and rises until it collides with an obstacle and generates multiple vortices. The rightmost column shows visualizations of the velocity fields using the Winter colormap on the velocity norm; blue implies slow or zero motion and green implies faster motion. In three dimensions streamlines are used to illustrate the evolution of fluid particles over time.

large number of streamlines to improve visibility, small features can appear as isolated strands.

7.2 Sub-grid Boundaries

Because our cut-cells always have full polynomial spaces our simulator can resolve complex flows along thin boundaries. This effect is especially visible in two dimensional results. Bernoulli's principle states that velocity through a tunnel is inversely proportional to cross sectional flux; as seen in Figure 9, the fluid velocity is indeed greater in the central thin gap and slows down further away. The Semi-circles example (Figure 10) is more extreme: the boundaries are much thinner than the grid while also having small gaps. The flow generated by our simulator not only runs through the thin pipes, but also correctly passes through the small gaps in the pipes.

Figure 9 illustrates our simulator's ability to easily handle diagonal boundaries, which axis-aligned Cartesian grid solvers often struggle with. This effectiveness is in part because, although our velocity degrees of freedom lie on faces, they are full velocities and not staggered flux samples.

Dragon. In a more challenging case, we can generate a stream of smoke through the dragon's mouth in Figure 11. Our simulator is robust against the noisy geometry of the mouth interior, which was an artifact of its original scanning process. VEM's stiffness matrices involve integrating over each of these noisy triangles, allowing it to capture the desired effect. The resulting flow navigates around fine features like the tongue and teeth while producing large-scale features like the two streams through the sides of the mouth.

7.3 Thin Features

Our simulator can resolve many thin details, including the intricate tendrils seen in *YeahRight* (Figure 1). Additional models with thin features are listed in Table 3.

7.4 Open and Non-Manifold Geometry

Mandoline creates volumetric elements from open geometry, so we can also simulate open or non-manifold geometry with little

Table 3. Statistics for our simulations. Note that Mandoline resolves all Self-Isects (i.e., self-intersections) using libigl, to yield consistent non-manifold triangle meshes before mesh generation.

Name	Features	Vertices	Faces	Grid	Cut-Cells	Cut-Faces	Particles
Torus	High genus	576	1152	50^3	119505	371457	1623616
Logs	Manifold, Closed	960	1860	50^3	114843	402251	1589545
Bunny	Manifold, Closed	34834	69663	50^3	119858	472045	1612111
Brucewick	Manifold, Closed	1431	2858	60^3	119486	442283	1633429
Hilbert	High genus, Thin features	19968	39936	50^3	32395	169988	1592720
Cow	Self-Isects, Thin features	2904	5804	50^3	120010	383585	1658280
Dragon	Self-Isects, Noisy	50000	100000	50^3	119863	508144	1639375
Plane	Open	4	2	50^3	119452	365714	1658280
Face	Open	25905	51712	50^3	119486	442283	1658280
Gummy-Jack	Self-Isects, Thin features	12243	24718	32^3	120393	416683	420971
YeahRight	High genus, Thin features	23324	47168	50^3	118937	429907	1656605
Monkey	Open	425	818	50^3	31072	99575	1658280
Vent	Open, Self-Isects	28	38	50^3	119408	367408	1645863
Crwth	Open, Self-Isects	5632	11401	32^3	30649	122523	1658348

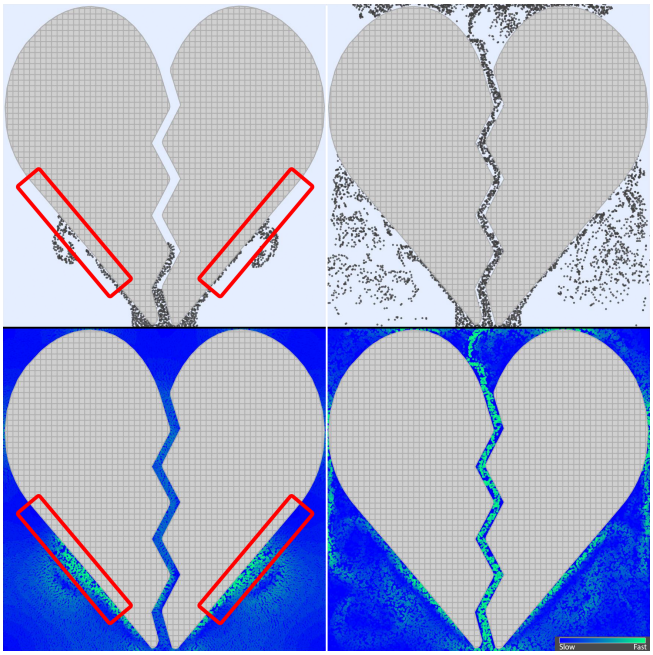


Fig. 9. Smoke initially rises along the walls of the heart and through the crack. The fluid maintains incompressibility by speeding up through the narrow tunnels in accordance with Bernoulli's principle. The red rectangles highlight some regions where fluid flows naturally along diagonal boundaries without aliasing artifacts.

extra effort. In fact, several of our prior examples also possess self-intersections (Table 3). The only notable difference in the computation of the VEM operators is the presence of cells in which both sides of a boundary face belong to a single cut-cell (i.e., "flaps" in Mandoline terminology). As we can see in the Face example (Figure 12) our simulator, most notably the pressure solver, is able to resolve distinct fluid flows on the separate sides of a thin surface. Because all of our

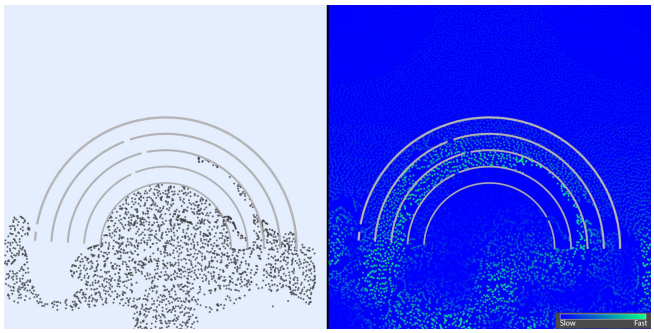


Fig. 10. Thin walls and narrow gaps are both supported by our pressure projection scheme.

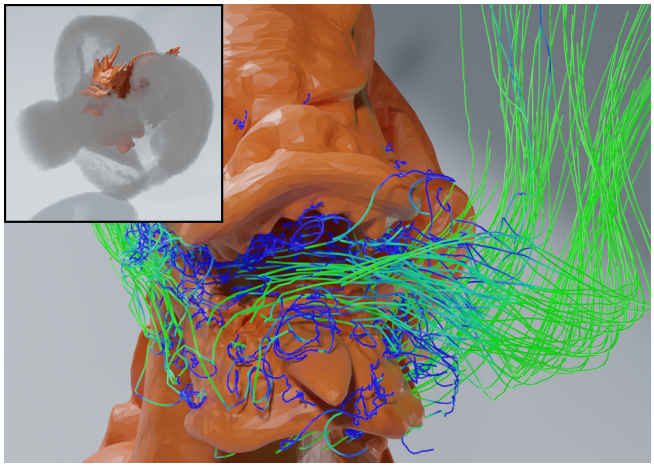


Fig. 11. Larger meshes with noisy geometry, such as the inside of this dragon's mouth, are handled by our simulator.

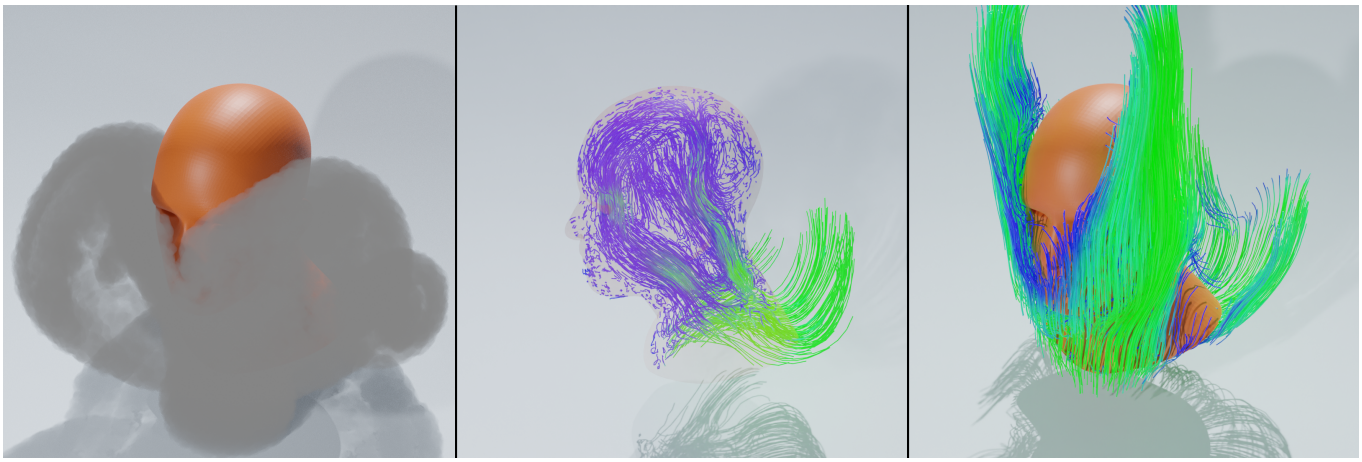


Fig. 12. The open face mesh generates two exterior plumes, flowing on either side of the face, and a third stream that flows through the interior, entering near the front and then returning out the back. The interior and exterior flows differ significantly despite having only an infinitesimal membrane separating them.

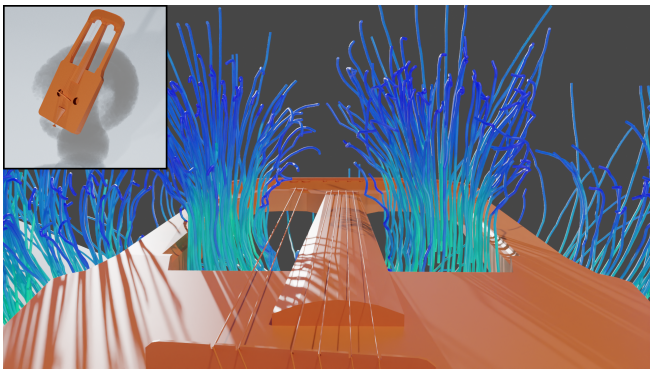


Fig. 13. Despite being far smaller than the simulation grid our simulator respects thin strings and flows around them.

computations involving boundary cells use boundary integrals, the contributions of the two coincident faces cancel each other out so we simply skip integrating those faces.

Particle advection can be slightly more challenging in such settings. Because our linear velocity fields cannot always *exactly* respect the complex geometry, some particles collide with boundaries and then slide right along them. With open geometries this is potentially problematic because floating point numerical precision issues can, in rare instances, result in a particle being misidentified as lying on the wrong side of a boundary, causing it to swap sides. This occurred at sharp corners of the geometry in only a handful of cases, so for our smoke renders we manually pruned the few smoke particles that leaked through, but did not do so in our velocity visualizations. Such leakages could be addressed with more elaborate, robust particle tracking and collision handling mechanisms, e.g., involving exact continuous collision detection and/or explicitly exploiting knowledge of cell connectivity and particle history. However, this issue did not substantially effect the observed behaviors.

7.5 Ease of Configuration

One of the key features of our method is that it is easy to create scenes with interesting behavior without much experience or specialized expertise in modeling for simulation. With our Monkey example (Figure 14), we simply removed some elements on the mouth and eyes of Blender's *Suzanne* model, as well as adding a smoke source under the eyes. Our simulator had no difficulty producing a plausible fluid flow with this open non-manifold mesh: air is naturally drawn in the mouth and exits the eyes as smoke.

Similarly, in the Vent example (Figure 15) we create an outer frame and attach some planes through it, without concern for how the planes penetrate the frame. Even with such a non-manifold geometry our simulator creates a flow that respects the blades.

7.6 Performance

Table 4 lists the timings for each component of our algorithm. Several of these procedures, such as the invocation of Mandoline and construction of operators like the Laplacian, divergence, and gradient operators, were performed only once at the beginning of the simulation. Because the vast majority of our elements are plain voxels we constructed a stencil for such a voxel once and duplicated it as needed. Table 4 shows that, with the exception of simple meshes like the Torus and Plane, VEMPIC's performance is dominated by the cost of pressure projection. Because both mapping particle data to meshes and advecting particles require identifying which cell each particle lies in we cached that information to reuse in both steps.

7.7 Geometric Fidelity Comparison to Mantaflow

We emulated our simulation setup in Blender's built-in smoke simulator Mantaflow [Thuerey and Pfaff 2018], with the aim of comparing the resolving power of our respective pressure solvers. To provide a reasonable comparison of our quadratic VEM pressure solve with Mantaflow's 7-point finite difference stencil we sought to match the

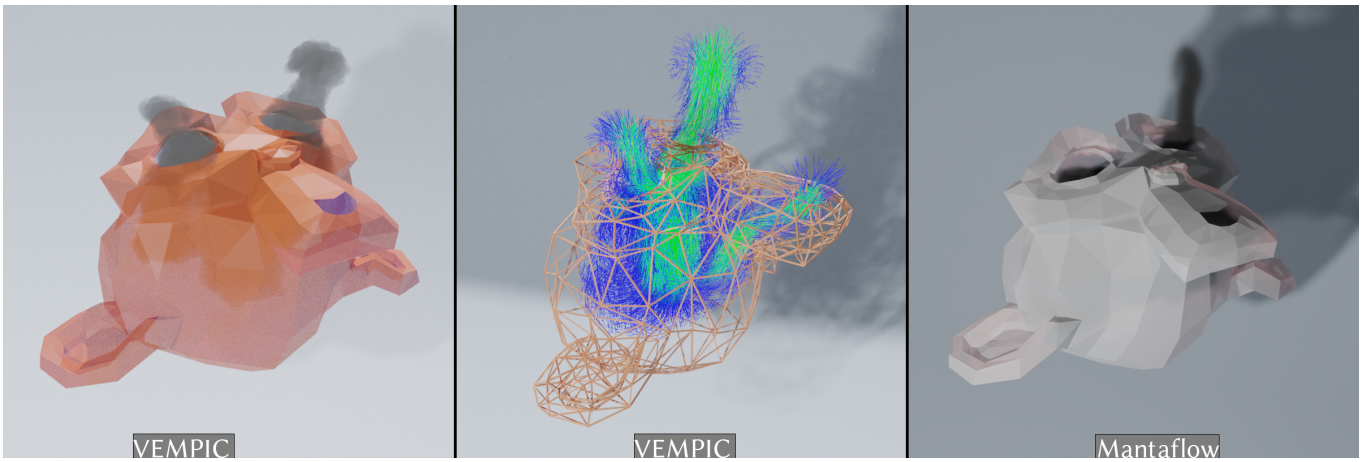


Fig. 14. We made holes through the eyes and mouth of Blender's *Suzanne* mesh and placed an emitter on the inside. Air naturally flows in through the mouth, is converted to smoke by the emitter region, and flows out through the eyes. Mantaflow, with its "Is Planar" setting only ejects a small stream out one eye.

Table 4. Average time for the various components of our simulations. Cell assignment is the identification of which cut-cell each particle lies in, as described in Section 6.2.

Name	Geometry	Operators	**Time/step**	**Cell Assignment/step**	**Particle→Mesh/step**	**Pressure/step**	**Advection/step**
Torus	1.901	18.108	27.264	4.094	13.456	6.879	3.536
Logs	3.744	20.024	227.909	4.242	15.179	205.239	4.149
Bunny	4.988	31.219	487.146	5.453	20.732	457.573	5.486
Brucewick	2.834	30.757	75.649	9.876	23.144	37.017	9.593
Hilbert	6.233	30.704	1218.726	4.496	19.632	1190.462	5.095
Cow	1.525	19.628	56.773	5.996	13.831	33.789	5.749
Dragon	5.835	49.696	1095.645	6.103	22.617	1063.198	6.396
Plane	1.908	18.334	24.466	4.082	13.158	4.423	3.451
Face	3.595	32.775	172.455	6.470	18.345	143.882	6.820
Gummy-Jack	3.740	24.132	253.786	7.644	16.550	225.392	8.471
YeahRight	3.323	24.747	82.401	5.902	17.740	55.310	5.945
Monkey	2.309	19.553	67.345	4.201	13.455	46.637	3.809
Vent	1.980	17.938	31.228	5.956	13.214	8.707	5.921
Crwth	1.047	28.777	794.121	4.222	15.249	771.729	3.724

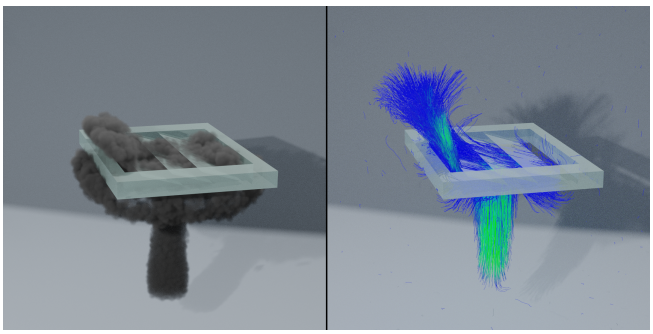


Fig. 15. This vent has both open faces (the blades) and self-intersections (where the blades meet the frame), but our simulator creates a natural flow.

number of degrees of freedom. This can be computed by using Table 2 and the number of cut-faces and cut-cells in our cut-cell mesh. Although MantaFlow is noticeably faster than our simulator, due to its simpler boundary treatment, sparser matrices, and significant optimizations (e.g., matrix-free multigrid solver, specialized matrix structures, etc.), we posit that with the same amount of data it resolves far less of the boundary than our simulator. We emphasize that our method uses particles to track smoke density, rather than MantaFlow's semi-Lagrangian advection of a coarse density grid, so our smoke also appears visually better resolved. However, this aspect is largely incidental; the important difference is the distinct flow structure induced by our pressure solver.

To test the two methods, we constructed a sequence of perpendicular logs of different widths at different elevations to observe at which resolution the simulators would detect and respond to features. With our simulator configured with a 50^3 grid and Mantaflow

Table 5. We used the Logs example of Figure 16 to test the effect of grid resolution on the number of degrees of freedom. "**L** NNZ" refers to the number of nonzeros in the system matrix. Interestingly, using a 60^3 grid was faster than using a 50^3 grid in this case; increasing the grid resolution decreased the geometric complexity of the cut-cells, making it easier to find valid pressure solutions.

Grid	Vertices	Cells	Faces	L NNZ	Time/step (s)
10^3	6479	984	9249	2636724	3.909
20^3	19890	7626	35900	8373024	8.454
30^3	44474	26293	97812	11454579	42.952
40^3	89503	63917	216259	22244596	71.465
50^3	156017	122793	402251	38502088	227.909
60^3	255858	213209	682882	61749726	172.644

configured with a corresponding 140^3 grid (at $\approx$ 2.3 seconds per frame), we noted quite different behavior. While our smoke both deflects off of and wraps around the logs, Mantaflow's smoke flows straight up or parallel to the logs, which shows its bias toward axis-aligned directions.

We also compared our simulator (at 50^3) with Mantaflow (at 90^3 and 0.48 seconds per frame) on our punctured monkey geometry (Figure 14) with the "Is Planar" option to indicate Non-manifold geometry. Unlike our simulator, which detected the three holes and generated flows through them, the Mantaflow smoke mostly remained trapped in the skull except for a small stream that flowed straight out of a single hole. Although our input grid resolutions are somewhat low, cut cells dramatically increase our method's ability to resolve flow near object boundaries and our quadratic pressure solver gives more resolving power further from the object. Other methods, even with increased grid resolution would have issues with open / nonmanifold / slender boundaries without significant additional effort and/or preprocessing.

7.8 Grid Resolution

Changing grid resolution does not always have monotonic effects when combining cut-cell meshes and VEM. On the one hand, increasing grid resolution leads to a larger system matrix, which generally takes longer to solve. On the other hand, coarse cut-cells might not have incompressible linear velocity fields that respect their boundaries, which affects the convergence rate of CG. Furthermore, with respect to each cut-cell, **L** is dense, so large cut-cells containing many triangles produce large dense blocks in the matrix. This results in higher memory consumption for cut-cell meshes with coarse grid resolutions, which also affects the performance of the sparse matrix multiplication required by CG. We therefore found that in some cases increasing grid resolution resulted in faster iterations. This is evident in Table 5 where the 60^3 system performed better than the 50^3 system. Additional simulations are shown in Figure 17 and Figure 18, as well as the accompanying video.

8 LIMITATIONS AND FUTURE WORK

Performance. Our priority in this work was to maximize flexibility and expressivity in handling complex boundaries, rather than raw speed, but performance remains a factor. The pressure solve requires 3 variables per face, and the number of total faces scales with the number of triangle faces and grid faces. Thus, extremely high resolution triangle meshes create a large number of degrees of freedom, and this issue is further exacerbated by cutting those triangles with the grid. Because each cell generates a local dense block in the Laplacian, the presence of too many triangles in a single cell can generate large dense matrices; in the extreme case, a single "cell" simulation results in a fully dense matrix. Interestingly, this reveals that multiplication with our Laplacian can sometimes be sped up by increasing the simulation resolution, suggesting that a judicious combination of our approach with an adaptive grid could be beneficial. We emphasize that octree-style spatial adaptivity is complementary (and orthogonal) to our use of Cartesian grid cut-cells and our VEM formulation, and thus the two could be naturally combined for improved performance. Furthermore, the storage of the Laplacian could be compressed by taking advantage of the identical stencil structure shared by the many full cubic voxels present in geometric cut-cell meshes, leading to additional speed-ups.

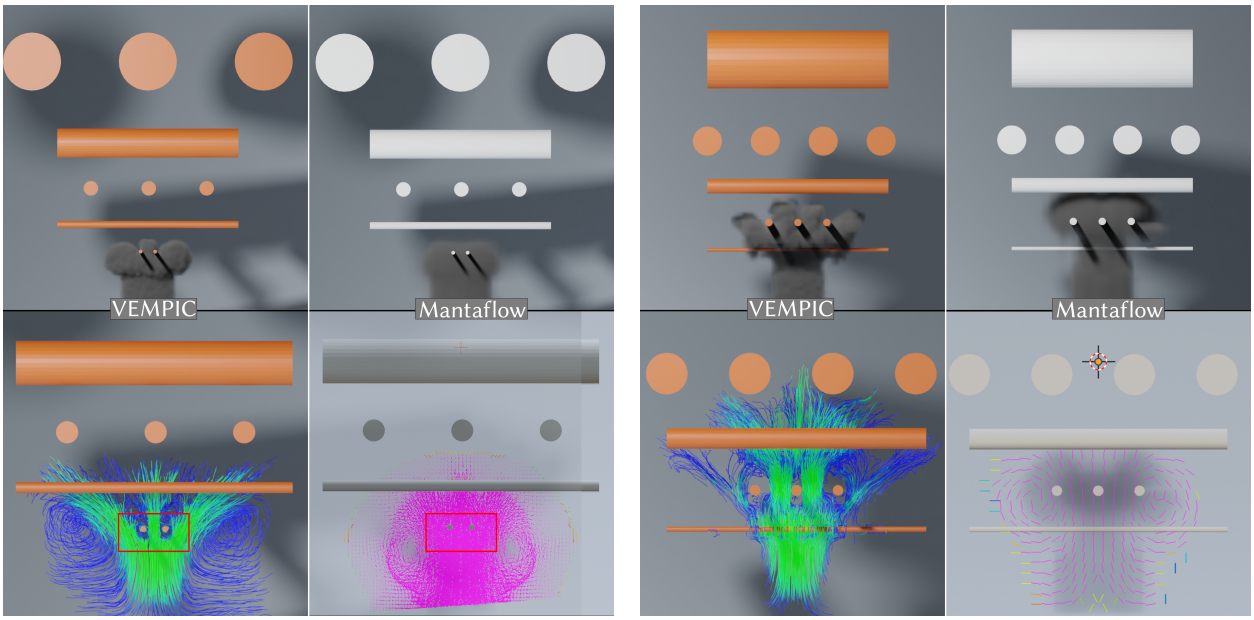
Further performance gains could be achieved through the development of a well-designed multigrid scheme to replace (or precondition) conjugate gradients. Similar to how we map data between particles and VEM degrees of freedom by computing m_j, different choices for estimating m_j could be used to compute appropriate restriction and prolongation operators.

Linear velocities. As the geometry in a single cut-cell can be arbitrarily complex, the existence of a nontrivial incompressible polynomial velocity field that satisfies free-slip boundary conditions is not guaranteed. This can result in single-cell vortices or particles colliding into walls. The emergence of single-cell vortices is a consequence of using linear velocities, so higher degree polynomials would be required to increase the fidelity of the simulator without increasing the grid resolution.

Due to the imprecision of floating point arithmetic, the interior queries of Section 6.2 can occasionally fail to return any cells. This tended to occur in cells with sharp corners. Because linear velocity fields can be insufficient to resolve flows around such boundaries, particles tend to repeatedly collide with them and increase the likelihood of such errors. As a fail-safe in such cases, we fall back to nearest-cell queries, but using higher order velocity fields to resolve these complex geometries would also be helpful.

Having inadequate polynomials in a complex cut-cell also affects the stiffness of our system. The velocity divergence near open boundaries tends to be poorly resolved by quadratic pressure projection, which increases the stiffness of our pressure solve. It would be interesting to add additional shape functions to handle these discontinuities. The development of a theoretical understanding of the precise relationship between mesh cell shape and the effectiveness of VEM remains a subject of ongoing investigation in computational mathematics [Sorgente et al. 2022], but could offer insights into how to best leverage cut-cells, grid adaptivity, and/or polynomial degree. While we observed variations in the rate of CG convergence for complex cut-cells, in practice we found our method to be stable for CFL numbers around 1, in line with comparable PIC/FLIP solvers.

Sampling. Our estimates for m_j could be improved: our approach is akin to a box filter, which behaves poorly when the particle distribution is biased or too sparse. Other filters could potentially



(a) Plumes Contact the First Row

(b) Plumes Contact the Second Row

Fig. 16. A phenomenological comparison between VEMPIC and Mantaflow's smoke simulator. When smoke hits the first row of logs VEMPIC's plume separates into three separate plumes, and upon reaching the second row of logs it splits once again. By contrast, Mantaflow's pressure solver does not identify the first row and barely notices one of the logs in the second row, as can be seen by the vertical velocity lines.

improve these estimates. Another alternative is to use polygonal quadratures or cubature [Kim and Delaney 2013] to allow for nodes to be explicitly placed on mesh faces.

We occasionally faced issues where tiny cut-cells would not contain particles, and in turbulent flows would report no velocities. This leads to discontinuities in the reconstructed velocity fields that appear as large divergences that must be resolved by the pressure solver. Ensuring a sufficiently high density of particles resolved these issues in practice. In the future, it would be helpful to develop resampling techniques or methods to gracefully elide tiny cells or the under-sampled degrees of freedom within them.

Another way to guarantee sufficient sampling might be to develop a compatible semi-Lagrangian advection scheme [Stam 1999], removing the dependency on the particle distribution, perhaps at the cost of less energetic flows.

Additional applications. Our contributions unlock a myriad of opportunities for future work. First, it would be interesting to explore more dynamic effects, such as free surface fluids, or to implement coupling between fluid and rigid or elastic materials. With no-slip boundary conditions it would be straightforward to assign the different regions of a cut-cell mesh to different material properties. Another advancement required for this direction would be the implementation of an elastic material model under the VEM formulation using cut-cells. Finally, we are curious to see the sorts of novel applications that our simulator may enable, such as the ability to provide feedback while prototyping aerodynamic geometries, either manually or procedurally.

9 CONCLUSION

We have presented a robust VEM-based algorithm for fluid simulation on cut-cell meshes that can, without any further intervention or preprocessing, resolve fine details such as thin tubes or infinitesimally thin walls, even when given pathological collision geometry. This algorithm builds upon the traditional PIC framework for fluid simulation by generalizing the standard particle-to-mesh and mesh-to-particle operators to the VEM degrees of freedom and polynomial space, respectively, as well as using VEM to perform a quadratic pressure projection. We have also shown how, although cut-cells are general polyhedra, cut-cell mesh processing can be made more tractable by taking advantage of the regular grid structure in a similar manner to hash tables. The result is a simulator robust to complex geometric inputs such as thin, open, or even non-manifold collision geometries that are not easily handled with prior approaches. Fundamentally, geometric modeling is often a difficult task and modeling *simulation-friendly* geometry is even harder. By relaxing the definition of simulation-friendly geometry, we hope that our new method will allow users to focus on creating their desired geometry, rather than on adapting or repairing it to satisfy the downstream simulator.

ACKNOWLEDGMENTS

This work is graciously supported by NSERC Discovery Grants (RGPIN-2017-05524 & RGPIN-2019-06895 & RGPIN-2021-02524), the Connaught Fund (503114), the Ontario Early Researchers Award (ER19-15-034), the European Research Council (714776 OPREP),

gifts from Adobe Research and Autodesk, and the Canada Research Chairs Program.

REFERENCES

2020. Intel oneTBB. *URl: http://github.com/oneapi-src/oneTBB* (2020).

Michael J Aftosmis, Marsha J Berger, and John E Melton. 1998. Robust and efficient Cartesian mesh generation for component-based geometry. *AIAA journal* 36, 6 (1998), 952–960.

Ryoichi Ando, Nils Thürey, and Chris Wojtan. 2013. Highly adaptive liquid simulations on tetrahedral meshes. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–10.

Vinicius C Azevedo, Christopher Batty, and Manuel M Oliveira. 2016. Preserving geometry and topology for fluid flows with thin obstacles and narrow gaps. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 97.

Christopher Batty, Florence Bertails, and Robert Bridson. 2007. A fast variational framework for accurate solid-fluid coupling. In *ACM Transactions on Graphics (TOG)*, Vol. 26. ACM, 100.

Christopher Batty and Robert Bridson. 2008. Accurate viscous free surfaces for buckling, coiling, and rotating liquids. (2008).

L Beirão da Veiga, Franco Brezzi, Andrea Cangiani, Gianmarco Manzini, L Donatella Marini, and Alessandro Russo. 2013. Basic principles of virtual element methods. *Mathematical Models and Methods in Applied Sciences* 23, 01 (2013), 199–214.

L Beirão da Veiga, Franco Brezzi, Luisa Donatella Marini, and Alessandro Russo. 2014. The hitchhiker's guide to the virtual element method. *Mathematical models and methods in applied sciences* 24, 08 (2014), 1541–1573.

Blender Online Community. 2018. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam. http://www.blender.org

Morten Bojsen-Hansen and Chris Wojtan. 2013. Liquid surface tracking with error compensation. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–13.

Landon Boyd and Robert Bridson. 2012. MultiFLIP for energetic two-phase fluid simulation. *ACM Transactions on Graphics (TOG)* 31, 2 (2012), 1–12.

Jeremiah U Brackbill, Douglas B Kothe, and Hans M Ruppel. 1988. FLIP: a low-dissipation, particle-in-cell method for fluid flow. *Computer Physics Communications* 48, 1 (1988), 25–38.

Franco Brezzi, Annalisa Buffa, and Konstantin Lipnikov. 2009. Mimetic finite differences for elliptic problems. *ESAIM: Mathematical Modelling and Numerical Analysis-Modélisation Mathématique et Analyse Numérique* 43, 2 (2009), 277–295.

F Brezzi and LD Marini. 2014. Virtual element and discontinuous Galerkin methods. In *Recent developments in discontinuous Galerkin finite element methods for partial differential equations*. Springer, 209–221.

Robert Bridson. 2007. Fast Poisson disk sampling in arbitrary dimensions. *SIGGRAPH sketches* 10 (2007), 1.

Yi-Lu Chen, Jonathan Meier, Barbara Solenthaler, and Vinicius C Azevedo. 2020. An extended cut-cell method for sub-grid liquids tracking with surface tension. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–13.

Alexandre Joel Chorin. 1968. Numerical solution of the Navier-Stokes equations. *Mathematics of computation* 22, 104 (1968), 745–762.

Pascal Clausen, Martin Wicke, Jonathan R Shewchuk, and James F O'brien. 2013. Simulating liquids and solid-liquid interactions with lagrangian meshes. *ACM Transactions on Graphics (TOG)* 32, 2 (2013), 1–15.

Gilles Daviet and Florence Bertails-Descoubes. 2016. A semi-implicit material point method for the continuum simulation of granular materials. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–13.

Blanca Ayuso de Dios, Konstantin Lipnikov, and Gianmarco Manzini. 2016. The nonconforming virtual element method. *ESAIM: Mathematical Modelling and Numerical Analysis* 50, 3 (2016), 879–904.

Fernando De Goes, Andrew Butts, and Mathieu Desbrun. 2020. Discrete differential operators on polygonal meshes. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 110–1.

Fernando De Goes, Corentin Wallez, Jin Huang, Dmitry Pavlov, and Mathieu Desbrun. 2015. Power particles: an incompressible fluid solver based on power diagrams. *ACM Trans. Graph.* 34, 4 (2015), 50–1.

Mathieu Desbrun and Marie-Paule Gascuel. 1996. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Computer Animation and Simulation'96*. Springer, 61–76.

Essex Edwards and Robert Bridson. 2012. A high-order accurate particle-in-cell method. *Internat. J. Numer. Methods Engrg.* 90, 9 (2012), 1073–1088.

Essex Edwards and Robert Bridson. 2014. Detailed water with coarse grids: Combining surface meshes and adaptive discontinuous galerkin. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 1–9.

Yu Fang, Ziyin Qu, Minchen Li, Xinxin Zhang, Yixin Zhu, Mridul Aanjaneya, and Chenfanfu Jiang. 2020. IQ-MPM: an interface quadrature material point method for non-sticky strongly two-way coupled nonlinear solids and fluids. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 51–1.

Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. 2001. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 15–22.

Yun Fei, Christopher Batty, Eitan Grinspun, and Changxi Zheng. 2018. A multi-scale model for simulating liquid-fabric interactions. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–16.

Florian Ferstl, Ryoichi Ando, Chris Wojtan, Rüdiger Westermann, and Nils Thuerey. 2016. Narrow band FLIP for liquid simulations. In *Computer Graphics Forum*, Vol. 35. Wiley Online Library, 225–232.

Nick Foster and Dimitri Metaxas. 1996. Realistic animation of liquids. *Graphical models and image processing* 58, 5 (1996), 471–483.

Uriel Frisch and Andrei Nikolaevich Kolmogorov. 1995. *Turbulence: the legacy of AN Kolmogorov*. Cambridge university press.

Chuyuan Fu, Qi Guo, Theodore Gast, Chenfanfu Jiang, and Joseph Teran. 2017. A polynomial particle-in-cell method. *ACM Transactions on Graphics (TOG)* 36, 6 (2017), 1–12.

Steven Gagniere, David Hyde, Alan Marquez-Razon, Chenfanfu Jiang, Ziheng Ge, Xuchen Han, Qi Guo, and Joseph Teran. 2020. A hybrid Lagrangian/Eulerian collocated velocity advection and projection method for fluid simulation. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 1–14.

Eran Guendelman, Andrew Selle, Frank Losasso, and Ronald Fedkiw. 2005. Coupling water and smoke to thin deformable and rigid shells. *ACM Transactions on Graphics (TOG)* 24, 3 (2005), 973–981.

Gaël Guennebaud, Benoit Jacob, et al. 2010. Eigen. *URl: http://eigen. tuxfamily. org* 3 (2010).

Ken Hayami. 2018. Convergence of the Conjugate Gradient Method on Singular Systems. https://doi.org/10.48550/ARXIV.1809.00793

Anil Nirmal Hirani. 2003. *Discrete exterior calculus*. Ph.D. Dissertation. California Institute of Technology.

Yuanming Hu, Yu Fang, Ziheng Ge, Ziyin Qu, Yixin Zhu, Andre Pradhana, and Chenfanfu Jiang. 2018. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–14.

David AB Hyde and Ronald Fedkiw. 2019. A unified approach to monolithic solid-fluid coupling of sub-grid and more resolved solids. *J. Comput. Phys.* 390 (2019), 490–526.

Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. 2015. The Affine Particle-in-Cell Method. *ACM Trans. Graph.* 34, 4, Article 51 (July 2015), 10 pages. https://doi.org/10.1145/2766996

Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. 2007. Harmonic Coordinates for Character Articulation. *ACM Trans. Graph.* 26, 3 (July 2007), 71–es. https://doi.org/10.1145/1276377.1276466

Peter Kaufmann, Sebastian Martin, Mario Botsch, Eitan Grinspun, and Markus Gross. 2009b. Enrichment textures for detailed cutting of shells. In *ACM SIGGRAPH 2009 papers*. 1–10.

Peter Kaufmann, Sebastian Martin, Mario Botsch, and Markus Gross. 2009a. Flexible simulation of deformable models using discontinuous galerkin fem. *Graphical Models* 71, 4 (2009), 153–167.

Theodore Kim and John Delaney. 2013. Subspace Fluid Re-Simulation. *ACM Trans. Graph.* 32, 4, Article 62 (jul 2013), 9 pages. https://doi.org/10.1145/2461912.2461987

Theodore Kim, Nils Thürey, Doug James, and Markus Gross. 2008. Wavelet turbulence for fluid simulation. *ACM Transactions on Graphics (TOG)* 27, 3 (2008), 1–6.

Dan Koschier, Jan Bender, and Nils Thuerey. 2017. Robust eXtended finite elements for complex cutting of deformables. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–13.

William E Lorensen and Harvey E Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *ACM siggraph computer graphics* 21, 4 (1987), 163–169.

Chaoyang Lyu, Wei Li, Mathieu Desbrun, and Xiaopei Liu. 2021. Fast and Versatile Fluid-Solid Coupling for Turbulent Flow Simulation. (2021).

Sebastian Martin, Peter Kaufmann, Mario Botsch, Martin Wicke, and Markus Gross. 2008. Polyhedral finite elements using harmonic basis functions. In *Computer graphics forum*, Vol. 27. Wiley Online Library, 1521–1529.

Michael Mengolini, Matías F Benedetto, and Alejandro M Aragón. 2019. An engineering perspective to the virtual element method and its interplay with the standard finite element method. *Computer Methods in Applied Mechanics and Engineering* 350 (2019), 995–1023.

Olivier Mercier, Cynthia Beauchemin, Nils Thuerey, Theodore Kim, and Derek Nowrouzezahrai. 2015. Surface turbulence for particle-based liquid simulations. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 1–10.

Marek Krzysztof Misztal, Kenny Erleben, Adam Bargteil, Jens Fursund, Brian Bunch Christensen, Jakob Andreas Bærentzen, and Robert Bridson. 2013. Multiphase flow of immiscible fluids on unstructured moving meshes. *IEEE transactions on visualization and computer graphics* 20, 1 (2013), 4–16.

Nicolas Moës, John Dolbow, and Ted Belytschko. 1999. A finite element method for crack growth without remeshing. *International journal for numerical methods in engineering* 46, 1 (1999), 131–150.

Neil Molino, Zhaosheng Bao, and Ron Fedkiw. 2004. A virtual node algorithm for changing mesh topology during simulation. *ACM Transactions on Graphics (TOG)* 23, 3 (2004), 385–392.
Joe J Monaghan. 1992. Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics* 30, 1 (1992), 543–574.
Patrick Mullen, Keenan Crane, Dmitry Pavlov, Yiying Tong, and Mathieu Desbrun. 2009. Energy-preserving integrators for fluid animation. *ACM Transactions on Graphics (TOG)* 28, 3 (2009), 1–8.
Michael B. Nielsen, Konstantinos Stamatelos, Morten Bojsen-Hansen, Duncan Brinsmead, Yannick Pomerleau, Marcus Nordenstam, and Robert Bridson. 2018. A Collocated Spatially Adaptive Approach to Smoke Simulation in Bifrost. In *ACM SIGGRAPH 2018 Talks* (Vancouver, British Columbia, Canada) *(SIGGRAPH '18)*. Association for Computing Machinery, New York, NY, USA, Article 77, 2 pages. https://doi.org/10.1145/3214745.3214749
Tobias Pfaff, Nils Thuerey, and Markus Gross. 2012. Lagrangian vortex sheets for animating fluids. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 1–8.
Daniel Ram, Theodore Gast, Chenfanfu Jiang, Craig Schroeder, Alexey Stomakhin, Joseph Teran, and Pirouz Kavehpour. 2015. A material point method for viscoelastic fluids, foams and sponges. In *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 157–163.
Andrew Selle, Ronald Fedkiw, Byungmoon Kim, Yingjie Liu, and Jarek Rossignac. 2008. An unconditionally stable MacCormack method. *Journal of Scientific Computing* 35, 2 (2008), 350–371.
Nicholas Sharp and Keenan Crane. 2020. A laplacian for nonmanifold triangle meshes. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 69–80.
Eftychios Sifakis, Kevin G Der, and Ronald Fedkiw. 2007a. Arbitrary cutting of deformable tetrahedralized objects. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 73–80.
Eftychios Sifakis, Tamar Shinar, Geoffrey Irving, and Ronald Fedkiw. 2007b. Hybrid simulation of deformable solids. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 81–90.
Funshing Sin, Adam W Bargteil, and Jessica K Hodgins. 2009. A point-based method for animating incompressible flow. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics symposium on computer animation*. 247–255.
Tommaso Sorgente, Silvia Biasotti, Gianmarco Manzini, and Michela Spagnuolo. 2022. The role of mesh quality and mesh quality indicators in the virtual element method. *Advances in Computational Mathematics* 48, 1 (2022), 1–34.
Jos Stam. 1999. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. 121–128.
Alexey Stomakhin, Craig Schroeder, Lawrence Chai, Joseph Teran, and Andrew Selle. 2013. A material point method for snow simulation. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–10.
Deborah Sulsky, Zhen Chen, and Howard L Schreyer. 1994. A particle method for history-dependent materials. *Computer methods in applied mechanics and engineering* 118, 1-2 (1994), 179–196.
Michael Tao, Christopher Batty, Eugene Fiume, and David IW Levin. 2019. Mandoline: robust cut-cell generation for arbitrary triangle meshes. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–17.
Yun Teng, David IW Levin, and Theodore Kim. 2016. Eulerian solid-fluid coupling. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 1–8.
Nils Thuerey and Tobias Pfaff. 2018. MantaFlow. *http://mantaflow.com.*
Ty Trusty, Honglin Chen, and David I.W. Levin. 2021. The Shape Matching Element Method: Direct Animation of Curved Surface Models. *ACM Transactions on Graphics* (2021). https://doi.org/10.1145/3450626.3459772
Ingo Wald, Sven Woop, Carsten Benthin, Gregory S Johnson, and Manfred Ernst. 2014. Embree: a kernel framework for efficient CPU ray tracing. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 1–8.
Omar Zarifi and Christopher Batty. 2017. A positive-definite cut-cell method for strong two-way coupling between fluids and deformable bodies. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (Los Angeles California). ACM, New York, NY, USA.
Yongning Zhu and Robert Bridson. 2005. Animating sand as a fluid. *ACM Transactions on Graphics (TOG)* 24, 3 (2005), 965–972.

A DERIVATION OF G_c

The entries of $\mathbf{G}$ can be directly extracted by a few manipulations using Green's first theorem:

$$\int_\Omega (\nabla\psi_i)\cdot(\nabla\psi_j) = \int_{\partial\Omega}(N\cdot\nabla\psi_i)\psi_j - \int_\Omega(\Delta\psi_i)\psi_j \tag{37}$$

$$= \int_{\partial\Omega}\sum_k \xi^\nabla_{k,i} g_k\psi_j - \int_\Omega\sum_\ell \xi_{\ell,i} g_\ell\psi_j \tag{38}$$

$$= \sum_k \xi^\nabla_{k,i}\frac{|c^k|}{|c^k|}\int_{c^k} g_k\psi_j - \sum_\ell \xi_{\ell,i}\frac{|c^\ell|}{|c^\ell|}\int_{c^\ell} g_\ell\psi_j \tag{39}$$

$$= \sum_k \xi^\nabla_{k,i}|c^k| m_k(\psi_j) - \sum_\ell \xi_{\ell,i}|c^\ell| m_l(\psi_j), \tag{40}$$

where ξ^∇ and ξ are the coefficients for $N\cdot\nabla\psi_i$ and $\Delta\psi_i$, respectively, in terms of the g_k, g_ℓ from the $\mathcal{V}$ basis. After applying integration by parts to determine Equation (37) the derivatives of ψ_i are written out in terms of polynomial bases g_k, g_ℓ. The terms are then rearranged to match the form of Equation (8) so the g_k, g_ℓ are absorbed into m_k, m_l to match Equation (26).

The entries of $\mathbf{G}$ are therefore these coefficients $\xi^\nabla_{k,i}|c^k|$ and $-\xi_{\ell,i}|c^\ell|$ for boundary face ϕ_k and cell ϕ_ℓ respectively.

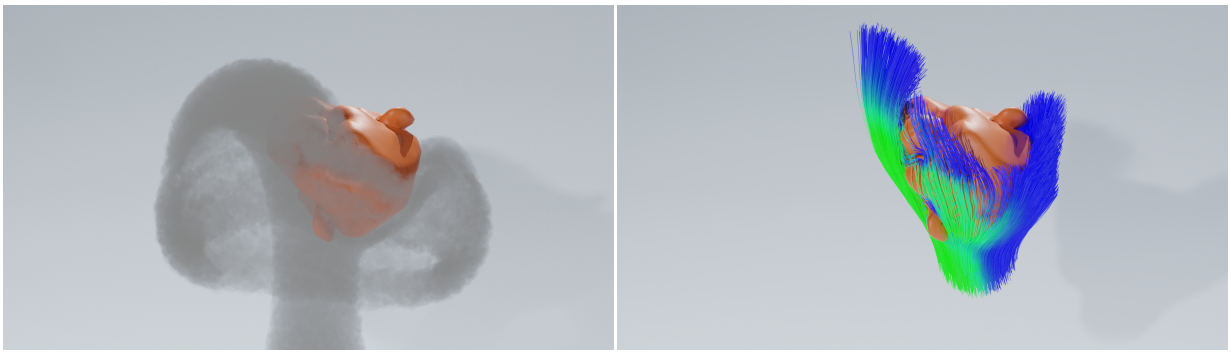
Implementing ξ is relatively straightforward because the basis used for the g_ℓ will be ψ_j up to order $k-2$; however, implementing the boundary ξ^∇ will require further elaboration, because each boundary face actually needs its own polynomial function space. We cannot define g_k as the restriction of ψ_j to the boundary face c_k because two adjacent cells might share c_k on their boundary and the restrictions would not be compatible with one another. The definition of the boundary g_k, and therefore ξ^∇, must be specified in terms of polynomial function spaces defined intrinsically on the boundaries of cells, which are discussed in Section 6.4.

(a) Torus — Rising smoke funnels through a small hole.

(b) Plane — A simple simulation of an open surface. Although the bottom side's flow moves quite quickly the flow is nearly static on the top.

(c) Hilbert Curve — Even though the gaps are around two grid cells wide we obtain many small plumes and vortices.

Fig. 17. Some additional simulation examples that were not explicitly discussed in the main text.



(a) Brucewick — The smoke flows around the small bumpy features of this face.

(b) Cow — The smoke flows around the thin legs and around the cow's belly. This mesh has some self-intersections near the tail.

(c) Gummy-Jack — Although the ears are thin and intersect with the face, plausible smoke plumes are generated.

Fig. 18. Some additional simulation examples that were not explicitly discussed in the main text.