We propose to augment standard grid-based fluid solvers with pointwise incompressible velocity interpolation, thereby ensuring exact incompressibility down to the sub-cell level. Our method takes as input a discretely divergence-free velocity field generated by a staggered grid pressure projection, and first recovers a corresponding discrete vector potential. Instead of solving a costly vector Poisson problem for the potential, we develop a fast parallel sweeping strategy to find a candidate potential and apply a gauge transformation to enforce the Coulomb gauge condition and thereby make it numerically smooth. Interpolating this discrete potential generates a pointwise vector potential whose analytical curl is a pointwise incompressible velocity field. Our method further supports irregular solid geometry through a use of level set-based cut-cells and a novel Curl-Noise-inspired potential ramping procedure at the heart of our approach is discretely divergence-free: \( \nabla \cdot \mathbf{u} = 0 \). Popular staggered grid-based schemes rely on this assumption, using finite difference or finite volume ideas, combined with Lagrangian or semi-Lagrangian advection methods, to transform the continuous incompressible flow equations into discrete, computable algorithms [Bridson 2015]. This fertile mathematical soil has sprouted diverse numerical tools for visual simulation of drifting cigarette smoke, coiling honey, crashing ocean waves, and more [Enright et al. 2002; Fedkiw et al. 2001; Larionov et al. 2017], which are widely integrated into industrial software like Houdini, Maya, and Blender. Yet despite the widespread use of spurious sources or sinks, and yield superior particle distributions over time.

Additional Key Words and Phrases: divergence-free, stream function, vector potential, velocity interpolation, advection

ACM Reference Format:
Jumyung Chang, Ruben Partono, Vinicius C. Azevedo, and Christopher Batty. 2022. Curl-Flow: Boundary-Respecting Pointwise Incompressible Velocity Interpolation for Grid-Based Fluids. ACM Trans. Graph. 41, 6, Article 243 (December 2022), 21 pages. https://doi.org/10.1145/3550454.3555498

1 INTRODUCTION

The assumption of incompressibility is pervasive in computer animation of fluids. Since compressive effects are imperceptible in many (but of course not all) visually relevant liquid and gas scenarios, neglecting fast-moving compression waves is often justifiable in practice and yields significant efficiency gains. Mathematically, incompressibility implies that the fluid velocity field \( \mathbf{u} \) should be divergence-free: \( \nabla \cdot \mathbf{u} = 0 \). Popular staggered grid-based schemes rely on this assumption, using finite difference or finite volume ideas, combined with Lagrangian or semi-Lagrangian advection methods, to transform the continuous incompressible flow equations into discrete, computable algorithms [Bridson 2015]. This fertile mathematical soil has sprouted diverse numerical tools for visual simulation of drifting cigarette smoke, coiling honey, crashing ocean waves, and more [Enright et al. 2002; Fedkiw et al. 2001; Larionov et al. 2017], which are widely integrated into industrial software like Houdini, Maya, and Blender. Yet despite the widespread use of
We exploit this relationship in the discrete and continuous settings. Under a finite volume approach, the discrete velocity components, stored at cell face midpoints, will indeed satisfy discrete incompressibility: the sum of discrete fluxes across each cell’s boundary is zero. However, interpolation is often required to provide pointwise velocity values everywhere in the simulation domain, to support popular Lagrangian or semi-Lagrangian discretizations of advection for passive tracer particles, density, velocity, temperature, and so on [Jiang et al. 2015; Stam 1999; Zhu and Bridson 2005]. Applying basic polynomial interpolants on discretely incompressible grid data affords no guarantee that the interpolated velocity fields will be pointwise analytically incompressible, and in practice they are not.

Spurious compressibility has non-negligible implications. Advecting particles through vector fields with artificial sources and sinks damages volume conservation and causes uniformly distributed particles to clump and spread. Alternatively, reducing discretization error by significantly increasing grid resolution fails to address the root cause and is too costly regardless: simulation time typically scales at least quartically with grid resolution. Irregular solid boundaries further exacerbate the issue because naive grid-based interpolants are essentially oblivious to obstacles; obstacle-aware cut-cell interpolants [Azevedo et al. 2016] improve the no-normal-flow enforcement at the cost of worsening compression artifacts.

To guarantee an analytically divergence-free interpolated velocity field by construction, we instead form the velocity field \( \mathbf{u} \) from the curl \( (\nabla \times) \) of a second vector field, \( \psi \), known as a vector potential:

\[
\mathbf{u} = \nabla \times \psi .
\]

In 2D, \( \psi \) reduces to a scalar stream function, \( \psi \). Incompressibility is thus enforced by a basic vector calculus identity,

\[
\nabla \cdot \mathbf{u} = \nabla \cdot (\nabla \times \psi ) = 0 .
\]

We exploit this relationship in the discrete and continuous settings to ensure pointwise incompressibility through three steps:

1. Given discretely incompressible fluid velocity values, construct corresponding discrete vector potential values.
2. Interpolate the discrete vector potential values to yield a pointwise vector potential field.
3. Take the analytical curl of the interpolated vector potential field to yield a pointwise incompressible velocity field.

Bao et al. [2017] first proposed this three-step recipe for deriving discrete delta functions in an immersed boundary method for weakly coupled fluid-structure interaction in computational fluid dynamics. However, their method is limited to perfectly uniform rectangular domains (i.e., no irregular solids) with periodic boundaries. Even so, their potential reconstruction approach requires the solution of a costly vector Poisson problem, making it several times more expensive than pressure projection in a standard fluid solver.

We extend this conceptual framework to be practical for animation applications in two and three dimensions by overcoming several challenges, including enabling much more efficient recovery of discrete vector potentials, enforcing exact no-flow or prescribed flux boundary conditions for exterior boundaries, and supporting irregular cut-cell obstacles. Our proposed Curl-Flow framework offers the following technical contributions:

- An efficient parallel sweeping algorithm to recover velocity-consistent discrete vector potentials on uniform and level set-based cut-cell grids;
- A gauge correction strategy to make the candidate vector potential well-suited to interpolation (i.e., numerically smooth) using a single fast, scalar Poisson solve;
- The design of an efficient, low-order vector potential interpolant that efficiently provides continuous and pointwise incompressible velocities in free space;
- A new additive ramping strategy for improved free-slip boundaries in the “Curl-Noise” method of Bridson et al. [2007];
- The application of this ramping method during velocity interpolation to enforce divergence-free, exact no-normal-flow conditions on axis-aligned exterior domain boundaries and interior irregular static obstacles in both 2D and 3D.

2 RELATED WORK

2.1 Within Computer Graphics

2.1.1 Recovering Discrete Vector Potentials. A few vector potential-based Eulerian/hybrid solvers have been proposed in the fluid animation literature, but most operate solely in the discrete realm and require solving a costly vector Poisson equation. Elcott et al. [2007] solved a vector Poisson problem to recover the vector potential from vorticity with a simplicial discretization of the vorticity equation. Ando et al. [2015] developed a vector Poisson-based alternative to the standard grid-based pressure projection. Notably, Ando et al. [2015] were the first to suggest employing vector potential-based velocity interpolation, in their discussion of future work. For an ideal Cartesian grid with only axis-aligned boundaries and certain choices of boundary conditions, the vector Poisson system can be decoupled into three scalar Poisson problems for efficiency; Sato et al. (likely) used this approach to recover the vector potential from a velocity field in fluid control applications [Sato et al. 2021, 2015]. Our proposed discrete vector potential reconstruction strategy offers a significantly more efficient solution, finding the desired potential at essentially the cost of a single scalar Poisson solve if the input velocity field is divergence-free, or two otherwise.

For 2D flow visualization, Biswas et al. [2016] recovered the discrete (scalar) stream function from velocity data and used marching squares to construct streamlines. They proposed an axis-based sweeping approach for 2D stream function recovery; we generalize this idea to irregular cut-cell boundaries and 3D vector potentials.

Relatedly, discrete vector field (Hodge) decomposition techniques have long been of interest in graphics [Tong et al. 2003]. Recently, more elaborate five-component discrete decompositions were considered by Poelke and Polthier [2016] and Zhao et al. [2019] for vector fields over triangulated surfaces and tetrahedral volumes, respectively, including various gauges and boundary conditions. These approaches determine the vector potential directly from the vector field via a vector Poisson solve, and in the five-component case, further require solving eigenproblems.
2.1.2 **Pointwise Divergence-Free Fields.** Various procedural techniques have used pointwise divergence-free velocity fields. Stam and Fiume [1993] first explored this idea for gaseous phenomena, treating incompressibility by restricting cross-spectral density functions in the Fourier domain. Curl-Noise [Bridson et al. 2007] uses a continuous vector potential to design animated divergence-free vector fields and the related scheme of DeWolf [2006] uses the cross-product of two gradient fields. Divergence-free sub-grid turbulence models subsequently built on these ideas [Kim et al. 2008; Schechter and Bridson 2008]. Incidentally, Schechter and Bridson [2008] also used (but did not describe) a divergence-free hybrid constant-linear velocity interpolant (see their supplemental video at time 2:18-2:27, left), which we show can be derived from bilinear stream function interpolation. This is the only prior instance of divergence-free grid-based interpolation in computer animation we know of. Pointwise divergence-free vector fields have also been used in geometry processing for volume-preserving modeling [Von Funck et al. 2006] and shape interpolation [Eisenberger et al. 2018].

2.1.3 **Divergence-Free Simulators.** Lagrangian vorticity-based simulation methods also employ a secondary vector variable to construct analytically divergence-free velocity fields; specifically, velocity is computed from vorticity via the Biot-Savart law. These methods come in many forms, including particles [Park and Kim 2005], filaments [Angelidis and Neyret 2005], and sheets [Brochu et al. 2012; Piaf et al. 2012]; closely related boundary integral/element (surface-only) methods also offer divergence-free fields [Da et al. 2016]. Model-reduction methods based on Laplacian eigenfunctions offer pointwise divergence-free fields via analytical basis functions [Cui et al. 2021; De Witt et al. 2012], albeit in simple domains.

2.1.4 **Subdivision Schemes.** Subdivision schemes based on discrete exterior calculus consider how to preserve properties of discrete differential operators on triangulated surface meshes undergoing refinement [De Goes et al. 2016; Wang et al. 2006]. These papers focus on surfaces and do not consider volumetric uniform grids or polyhedral cut-cells. More fundamentally, they achieve pointwise incompressibility only in the limit of infinite refinement. It would be interesting to adapt these ideas to polyhedra.

2.1.5 **Enhanced Advection.** A wide array of fundamental improvements to advection have been proposed to go beyond basic (trilinear) interpolation and (semi-Lagrangian) advection schemes [Stam 1999], ranging from monotonic cubic interpolation [Fedkiw et al. 2001] to MacCormack advection [Selle et al. 2008] to variants of particle-in-cell schemes [Jiang et al. 2015; Zhu and Bridson 2005]. Such methods tend to focus on vorticity-preservation, higher order accuracy, or diffusion-reduction. Because they typically use the underlying interpolant as a black box, they are orthogonal, and often complementary, to our scheme, as we demonstrate for FLIP [Zhu and Bridson 2005].

2.2 **Outside Computer Graphics**

2.2.1 **Divergence-Free Finite Element Methods.** In applied math, a wide range of discontinuous Galerkin (DG) and other non-conforming finite element methods (FEM) have been developed to offer pointwise divergence-free fields, often by adopting carefully designed incompressible per-element basis functions (e.g., [Cockburn et al. 2004; Lehrenfeld and Schöberl 2016; Rhebergen and Wells 2018]). In particular, Maljaars et al. [2018] combined an exactly divergence-free Hybridized Discontinuous Galerkin scheme with a particle-in-cell method (in 2D, without obstacles) and demonstrated better particle distributions. However, this family of FEM methods tends to have a more complex implementation and do not naturally integrate with standard approaches in animation. Moreover, they possess inter-element field discontinuities by their very nature, which can be problematic for visual applications; Maljaars et al. [2018] observed that reducing the magnitude of these inter-element discontinuities helps to improve particle distributions. Guzmán and Neilan [2014] tackled the more challenging task of designing a fully conforming/continuous Stokes finite element method yielding divergence-free solutions on 3D tetrahedra. Doing so required a specialized finite element basis consisting of a combination of cubic polynomials and divergence-free rational functions, and due to the construction’s complexity the authors note that “practical significance of the proposed elements may be questionable”. In isogeometric analysis, Evans and Hughes [2013] developed an exactly divergence-free Navier-Stokes simulation framework based on geometrically mapped rectangular B-spline grids; approaches in this vein require complex mesh construction for non-trivial domains, in contrast to the simpler and more efficient cut-cell techniques often preferred in animation. In fact, all of the approaches above would necessitate replacing the entire Navier-Stokes simulator, while ours provides a convenient plug-in upgrade to the advection phase of industry-standard visual effects methods. Thus, somewhat closer to our approach are methods that post-process flow solver solutions to exactly recover a divergence-free velocity field. For example, Linke [2012] converts the solution of a staggered triangulated discretization into a strictly divergence-free field in terms of Raviart-Thomas elements. Similarly, Lederer et al. [2017] proposed a velocity reconstruction operator that maps discretely divergence-free fields of Taylor-Hood and mini elements to exactly divergence-free ones, and suggested using their method as a postprocessing of the discrete solution in the conclusion. However, like the DG methods discussed above, the resulting finite element spaces (i.e., velocity fields) are not continuous between elements.

2.2.2 **Direct Interpolation of Finite Volume Solutions.** In a computational fluid dynamics context, Jenny and colleagues [Jenny et al. 2001; Meyer and Jenny 2004] derived 2D divergence-free node-based velocity interpolants for uniform grids and demonstrated improved particle distributions in particle-in-cell schemes. In geodynamics, Wang et al. [2015] developed a 3D extension and Pusok et al. [2017] adapted it for face-based data by first averaging to nodes. These approaches augment nonlinear interpolants with corrective terms to satisfy pointwise incompressibility per cell. In astrophysics (magnetohydrodynamics), Balsara [2001, 2004] proposed very similar divergence-free vector field reconstruction strategies based on local piecewise fitting on each cell or tetrahedron; one presupposes a polynomial basis and uses the divergence-free condition to determine constraints. Balsara [2009] presented an extension of this approach to Cartesian grid WENO schemes. These approaches do not consider cut-cell geometries, and exhibit kinks between cells.
Conveniently, the approaches above do not require recovering a vector potential at all. However, the recovered fields exhibit pervasive discontinuities between cells, and to achieve a given order of accuracy or support a particular geometry, they must be developed each time on a case-by-case basis. Our choice to instead recover and exploit an explicit discrete vector potential makes it comparatively straightforward to derive generalizations of our framework to higher order accuracy, global continuity/smoothness, or different element shapes, using relatively standard interpolation techniques, including spline interpolants or mesh-free methods. It could also be fruitfully combined with fluid control tools that already rely on vector potentials [Sato et al. 2021, 2015].

Concurrent work by Schroeder et al. [2022] proposes a direct spline-based interpolation method for staggered regular grid velocity fields that yields pointwise divergence-free fields of different orders. Some variants of their method can also exactly interpolate the cell normal components. This method avoids recovering the vector potential, but is limited to uniform grids without obstacles.

An alternative meshless divergence-free interpolation strategy [Lowitzsch 2005; McNally 2011] exploits matrix-valued radial basis functions and properties of the vector Laplacian. While this method offers divergence-free fields, as a meshless point-based method it struggles to precisely enforce boundaries and, more critically, preliminary tests showed it to be massively expensive, since it uses a fully dense global solve to simultaneously determine coefficients for all of the radial basis functions. Even attempting to substitute local kernels for the method’s global kernels remains expensive: a large support radius (i.e., particle neighborhood) is needed to avoid local oscillations and thus the system matrix is still unattractively dense.

2.2.3 Vector Potentials and Gauge Conditions. The methods most similar to ours are those that use explicit vector potentials. Since the vector potential for a given velocity field is not unique, various gauge conditions are used to select a particular potential from this null space. The gauge choice provides one dimension along which to categorize the schemes discussed below. Another is whether they apply a local approach to find the vector potential, based on a cell-by-cell (possibly branching) traversal, or find a simultaneous global solution by solving a linear system. We straddle these categories: our parallel sweeping approach is a fast cell-by-cell scheme that yields an initial potential, which we correct with a relatively inexpensive scalar Poisson solve (via Discrete Sine Transform or DST) to ensure smoothness by enforcing the Coulomb gauge, \( \nabla \cdot \psi = 0 \). Furthermore, beyond our innovations in the uniform grid setting, our work is unique in considering solid geometry via cut-cells.

The Coulomb gauge condition picks out a unique smooth vector potential by enforcing that it be divergence-free; it is the most common choice and has been used in graphics (e.g., [Ando et al. 2015] among others.) The initial implicit gauge condition used in our 3D parallel sweeping method is conceptually similar to that of Ravu et al. [2016], who sought to directly construct spline-based vector potentials. Like us, they set the \( \varphi_z \) component to zero. However, their overall approach differs in that (a) their velocity and vector potential degrees of freedom are colocated at nodes, (b) they directly seek the coefficients of continuous cubic spline functions rather than edge-based discrete vector potential values (hence four times as many unknowns), and (c) compared to our fast parallel sweeping, they solve an expensive global linear system to find the many coefficients.

Another implicit gauge condition falls out from the tree-cotree approach, explored for edge-based finite element methods by Albanese and Rubinacci [1990] on grids and Manges and Cendes [1995] on meshes. This global approach determines a spanning tree of an edge-based grid or mesh that one can safely set to zero, in order to explicitly eliminate the null space of the matrix representing the discrete curl operator. This idea has natural connections to the traversal patterns of cell-by-cell approaches.

Bao et al. [2017] proposed a divergence-free interpolation strategy on uniform staggered grids that reconstructs a discrete vector potential field from velocities, to improve coupling and reduce volume conservation errors in the immersed boundary (IB) method [Peskin 2002]. (IB uses smeared delta functions for approximate coupling with solids, in contrast to our “sharp” cut-cells.) Similar to methods in graphics [Ando et al. 2015; Tong et al. 2003], they recover the vector potential under the Coulomb gauge using a vector Poisson system. They also assume periodic boundary conditions, which are undesirable for most graphics applications, and only suggest non-periodic domains as future work. Their periodic, obstacle-free setting decouples the vector problem into three scalar Poisson solves, which they treat via the fast Fourier transform (FFT). Our method recovers the same potential with an efficient sweep process followed by just one scalar Poisson solve (which can likewise be done via FFT on simple domains). We further support interior obstacles and closed domains with a fast DST-based solve. For interpolation, they use tensor-product B-splines, similar to ours. Casquero et al. [2018] also considered the immersed boundary method on uniform grids, but like Evans and Hughes [2013] and in contrast to Bao et al. [2017], they perform the entire simulation using divergence-free spline basis functions rather than post-processing staggered data.

Silberman et al. [2019] adopt a staggered uniform grid configuration and seek a discrete edge-based vector potential, similar to our setting, but for applications in electromagnetics. They propose a sequential, cell-by-cell flooding strategy which is more costly than our approach. Specifically, our axis-based sweeping safely assumes zero values in the \( z \)-component, which reduces the computation per cell, the number of unknowns (by a third), and the number of special cases, while enabling a fast parallel solution via 1D sweeps. Like us, they use a Poisson-based correction to apply the Coulomb gauge, but like Bao et al. [2017] they support only uniform grids, and they use infinite boundary conditions to allow solution by FFT. Silberman et al. [2019] additionally suggest an alternative “global linear algebra” approach that is loosely similar to the vector Poisson method of Ando et al. [2015] in that it both removes divergence and recovers a potential satisfying the Coulomb gauge; they note that it scales worse than their flood-then-correct approach.

In summary, our vector potential recovery approach is most closely related to the uniform staggered grid methods of Bao et al. [2017] and Silberman et al. [2019], but is more efficient and generalizes these ideas to the more flexible domain boundary conditions and irregular cut-cell solids favored in graphics applications.
We represent solids using node-based level sets, leading to simple
we build on a standard staggered grid-based fluid solver for the
wise vector field that is analytically divergence-free, while ideally
where \( A \)

\[ u \]

where \( q \) is any scalar (e.g., density) or vector (e.g., velocity) quantity to be advected. Common
advection discretizations use semi-Lagrangian [Stam 1999] or La-
grangian [Zhu and Bridson 2005] methods that trace trajectories
through the flow and require interpolation to query the velocity.

The input discrete velocity field \( u \) to be used for advection is
assumed to be discretely divergence-free due to the underlying
simulator’s incompressibility enforcement (i.e., pressure projection).
Discrete velocity components are placed at grid face centers, so the
discrete divergence-free condition is

\[ \sum_{\text{faces } f} A_f u_f = 0, \]

where \( A_f \) is face area and \( u_f \) is the face’s outward oriented normal
velocity. Given such discrete data \( u \), our task is to generate a point-
wise vector field that is analytically divergence-free, while ideally
being continuous, smooth, and boundary-respecting.

Bilinear interpolation applied to each staggered velocity compo-
nent independently is the simplest common interpolant, but the
resulting vector field’s analytical divergence is nonzero in general
(see Section 1 in supplemental material). Figure 2, left column, vi-
sualizes such a flow and its divergence; if one advects uniformly
sampled particles through this field (Figure 3, top row), a sink region
absorbs a large number of particles while many more cluster into
large and small rings. Unfortunately, neither higher order interpo-
ation (Figure 3, middle row) nor using node-based / colocated veloc-
ity data will resolve these issues, since those interpolants remain
divergence-oblivious. We use staggered multilinear interpolation as
our baseline direct velocity interpolant throughout the paper, unless
stated otherwise.

The middle and right columns in Figure 2 visualize the linear and
quadratic variants of our proposed approach, respectively, developed
in the next section. Since the linear case exhibits velocity kinks
we ultimately prefer quadratic, but neither flow contains sinks or
sources and their pointwise divergence fields are exactly zero.

Section 2.2.2 mentioned prior approaches that aim for divergence-
free interpolation using velocity data directly, rather than needing
intermediate potentials. In particular, Jenny et al. [2001] and Balsara
[2001] used the same staggered grid layout as us in 2D (and in prac-
tice yield identical fields to one another). These direct incompressible
velocity interpolants produce pointwise divergence-free velocities
within each cell, but kinks are visible between most cells (see Fig-
ure 4) due to their piecewise construction and use of the minmod
limiter [Jenny et al. 2001], unlike our (quadratic) Curl-Flow method.

Fig. 2. Pointwise Divergence Comparison: Left: Bilinear interpolation
of discretely divergence free velocity data on a 3 × 3 grid. The red box
in (a) contains a spurious sink. Middle: Linear Curl-Flow interpolation is
divergence-free, but has piecewise constant components that induce kinks.
Right: Quadratic Curl-Flow interpolation is divergence-free and smooth.

Fig. 3. Effect on Particle Distribution: Initially uniform particles advected
through a static 2D vector field on a 3 × 3 grid. Frames 1, 100, 200, and
300 are shown left to right. (a) With direct bilinear velocity interpolation,
particles become clustered, leaving large empty voids. (b) With higher order
direct velocity interpolation (monotonic cubic [Fedkiw et al. 2001; Fritsch
and Carlson 1980]), clustering and spreading remain significant. (c): With
quadratic Curl-Flow interpolation, particles remain better distributed.
We first describe our approach in 2D without obstacles. Velocity discretized on each edge, gives the relationship
\[
\mathbf{v} = \nabla \phi \cdot \mathbf{e}_{ij} + \nabla \psi \cdot \mathbf{n},
\]
denoted \( \psi \), called the stream function and denoted by the dot product simply determines the sign. Since \( e_{ij} \) is oriented, unit length, and matches \( \mathbf{n} \) up to a sign flip, the dot product simply determines the sign.

4 CURL-FLOW INTERPOLATION IN 2D

We first describe our approach in 2D without obstacles. Velocity \( \mathbf{u} \) has two components, \( u \) and \( v \), and the vector potential has one scalar component, \( \psi \), denoted \( \nabla \psi \). The relationship (1) simplifies to
\[
\nabla \phi = \frac{\partial \psi}{\partial y}, \quad \nabla \psi = -\frac{\partial \phi}{\partial x}.
\]
That is, \( \mathbf{u} \) is a 90° rotation of \( \nabla \psi \), denoted \( \nabla \psi^\perp \). To discretize, we place \( \psi \) samples at cell vertices (Figure 5) and assume \( \phi \) varies linearly on edges. Figure 5(c) illustrates two nodal \( \psi \) values and the normal component of velocity, \( \phi_n \), on a shared edge \( e_{ij} \) having unit tangent \( \mathbf{e}_{ij} \) and length \( l \). The gradient theorem,
\[
\int_{e_{ij}} \nabla \psi(r) \cdot d\mathbf{r} = \psi(x_j) - \psi(x_i),
\]
discretized on each edge, gives the relationship
\[
\frac{\psi_j - \psi_i}{l} = \nabla \psi \cdot \mathbf{e}_{ij} = \nabla \psi^\perp \cdot \mathbf{n} = \phi_n \mathbf{e}_{ij}^\perp.
\]
Since \( \mathbf{e}_{ij}^\perp \) is oriented, unit length, and matches \( \mathbf{n} \) up to a sign flip, the dot product simply determines the sign.

4.1 Uniform Grids in 2D

For a single uniform grid cell as in Figure 5(a), equation 7 leads to
\[
\begin{align*}
\psi_1 &= \frac{\psi_2 - \psi_0}{h}, & \psi_r &= \frac{\psi_4 - \psi_2}{h}, \\
\psi_b &= -\frac{\psi_1 - \psi_0}{h}, & \psi_e &= -\frac{\psi_3 - \psi_2}{h},
\end{align*}
\]
consistent with finite differences on (5). We seek \( \psi \) satisfying the given velocities. These equations have a 1D null space: constant offsets of all \( \psi_j \) do not change the velocities. (Physically, discrete incompressibility ensures one flux is the negated sum of the others, implying one equation is linearly dependent.) We select a unique solution by arbitrarily setting \( \psi_0 = 0 \). We determine the other \( \psi \) by traversing the cell’s edge graph, computing each subsequent \( \psi \) value from its predecessor on the edge and the edge’s velocity component. Discrete incompressibility ensures discrete integrability, i.e., looping back to \( \psi_0 \) gives a consistent result.

This graph traversal scheme applies likewise to a whole uniform grid of cells, thereby avoiding a costly global Poisson solve for \( \psi \) based on (2). Any ordering suffices, so we aim to maximize parallelism. As proposed by Biswas et al. [2016] and illustrated in Figure 6, we first set \( \psi_0 = 0 \), and sweep vertically to obtain all the \( \psi \) values at \( x = x_0 \) (i.e., \( \psi_1 = \psi_0 + h \alpha_0 \) and so on). We then sweep horizontally, in parallel, to obtain all remaining \( \psi \) values. This approach is compatible with any standard exterior domain boundary conditions (inflow/outflow/open/closed), since it only needs the velocity data.

4.1.1 Recovering discrete \( \psi \). For a single uniform grid cell as in Figure 5(a), equation 7 leads to
\[
\begin{align*}
\psi_1 &= \frac{\psi_2 - \psi_0}{h}, & \psi_r &= \frac{\psi_4 - \psi_2}{h}, \\
\psi_b &= -\frac{\psi_1 - \psi_0}{h}, & \psi_e &= -\frac{\psi_3 - \psi_2}{h},
\end{align*}
\]
consistent with finite differences on (5). We seek \( \psi \) satisfying the given velocities. These equations have a 1D null space: constant offsets of all \( \psi_j \) do not change the velocities. (Physically, discrete incompressibility ensures one flux is the negated sum of the others, implying one equation is linearly dependent.) We select a unique solution by arbitrarily setting \( \psi_0 = 0 \). We determine the other \( \psi \) by traversing the cell’s edge graph, computing each subsequent \( \psi \) value from its predecessor on the edge and the edge’s velocity component. Discrete incompressibility ensures discrete integrability, i.e., looping back to \( \psi_0 \) gives a consistent result.

This graph traversal scheme applies likewise to a whole uniform grid of cells, thereby avoiding a costly global Poisson solve for \( \psi \) based on (2). Any ordering suffices, so we aim to maximize parallelism. As proposed by Biswas et al. [2016] and illustrated in Figure 6, we first set \( \psi_0 = 0 \), and sweep vertically to obtain all the \( \psi \) values at \( x = x_0 \) (i.e., \( \psi_1 = \psi_0 + h \alpha_0 \) and so on). We then sweep horizontally, in parallel, to obtain all remaining \( \psi \) values. This approach is compatible with any standard exterior domain boundary conditions (inflow/outflow/open/closed), since it only needs the velocity data.

4.1.2 Interpolating \( \psi \). Interpolation of the grid \( \psi \) values provides an analytical stream function at every point. Applying the analytical curl, we obtain a velocity field which is pointwise divergence-free by construction. As a simple example, for bilinear interpolation (referring to the inset figure),
\[
\psi(x, y) = \text{lrep}(\text{lrep}(\psi_0, \psi_1), \beta(y), \alpha(x)),
\]
\[
u(x, y) = -\frac{\text{lrep}(\psi_0, \psi_1, \psi_2, \beta(y), \alpha(x))}{h} = \text{lrep}(\psi_0, \psi_1, \beta(y)),
\]
where \( \text{lrep}(a, b, t) = (1-t)a+tb \) and the rightmost equalities follow from (8).
zero because it exactly equals the finite difference divergence:
\[
\nabla \cdot \mathbf{u} = \frac{u_x - u_t + \psi_t - \psi_h}{h} = 0.
\]

The derivatives in the curl operator inherently induce a different polynomial degree per axis: here, velocity components are piecewise constant in one axis and piecewise linear in the other, causing undesired tangential discontinuities between cells (Figure 2, middle).

Fortunately, because we recovered the discrete \( \psi \), we can achieve velocity continuity simply by upgrading to a higher order interpolant (Figure 2, right). Although many choices are possible, we adopt quadratic dyadic B-spline kernels [Jiang et al. 2016; Steffen et al. 2008]; they are sufficiently smooth, their analytical curl is straightforward to derive, and they possess small stencils for efficiency. Bao et al. [2017] similarly used tensor-product B-splines when constructing divergence-aware Dirac delta functions.

For the special case of bilinear \( \psi \) interpolation on 2D uniform grids, (9) shows that the velocity interpolant reduces to a simple function of the input discrete \( u, v \) velocities, eliminating the need for explicit \( \psi \) values. This effect occurs because, in 2D, each velocity component is defined by a single partial derivative of \( \psi \) and the null space is a constant shift. However, deriving simple direct incompressible velocity interpolants in this way becomes unwieldy or impossible in the combined presence of cut-cell geometry, higher order interpolation, and especially 3D, where each velocity component depends on two vector potential components and the null space is nontrivial. We therefore prefer to explicitly recover \( \psi \).

4.2 Cut-Cell Obstacles in 2D

We next extend our scheme to support irregular solid obstacles.

4.2.1 Recovering discrete \( \psi \). Considering cut-cell solids during stream function recovery only requires accounting for truncated grid edges and non-axis-aligned "cut edges". Applying (7) we get an update rule for sweeping through such edges:
\[
\psi_{i+1} = \psi_i + \Delta_i \psi_j \left( \mathbf{e}_i \cdot \mathbf{n}_{i+1} \right).
\]

For the static obstacles we consider, the recovered \( \psi \) will be constant on the surface, i.e., the surface is an isocountour that particles slide along in a free-slip manner [Bridson et al. 2007]. Static obstacles also preserve the remarkable simplicity of our parallel sweeping method: nodal stream function values at the entry and exit point of a grid line passing through an obstacle are identical, so we can sweep through obstacles by assuming zero velocities inside.

4.2.2 Interpolating \( \psi \). Cut-cell solids cause the nodal \( \psi \) values (blue in the inset) to no longer lie in a convenient uniform grid, thereby complicating interpolation. One could use moving least squares (MLS) interpolation based on \( \psi \) data in some neighborhood, but MLS requires solving small dense linear systems and computing the necessary analytical derivatives is difficult [Huerta et al. 2004]. Instead we extrapolate \( \psi \) values to uniform grid nodes inside the solid (i.e., \( \psi_s \) in the inset) and use quadratic B-spline kernels as in the uniform grid case. This extrapolation can be done in various ways, such as using MLS or simply copying (e.g., from \( \psi_g \) to \( \psi_s \)), assuming discrete flux is zero on interior solid edges (dashed lines); we adopt the latter for simplicity. At the outer axis-aligned domain boundaries, the quadratic stencil is also missing data samples, so we extrapolate an extra uniform layer of \( \psi \) samples outside the domain. However, regardless of these various extrapolation and interpolation choices near boundaries, the desired pointwise no-normal-flow boundary condition is (so far) enforced only approximately; we propose a final correction below.

4.2.3 A Curl-Noise enhancement for exact 2D boundary enforcement.

To ensure that the interpolated \( \psi \) will be strictly constant along the polygonal boundary for static obstacles, we adapt ideas from the Curl-Noise method [Bridson et al. 2007]. That method modulates the existing \( \psi \) value near obstacles, forcing it to a constant zero value on the boundary by multiplying against a smooth ramp function based on boundary proximity. Defining \( d_0 \) as an influence radius (we use the grid width \( h \)), \( d(\cdot) \) as the distance to the surface, and \( \text{ramp}(\cdot) \) as a smooth ramp function (we use a classic smoothstep function), Bridson’s multiplicative correction to the ambient \( \psi \) is the product \( \psi'(x) = \alpha \psi(x) \), where \( \alpha = \text{ramp}(d(x)/d_0) \).

However, depending on the background \( \psi \), the appropriate target boundary value \( \psi_g \) is often not zero; arbitrarily choosing zero leads to dramatic spurious flow deviations (Figure 7). One can modify Bridson’s method to “ramp in” the nonzero target boundary value \( \psi_g \):
\[
\psi'(x) = \alpha \psi(x) + (1 - \alpha) \psi_g.
\]

Unfortunately, the initial multiplication still damages the normal derivatives of \( \psi \). Since \( \mathbf{u} = \nabla \psi \) implies \( \mathbf{u} \cdot \mathbf{t} = \nabla \psi \cdot \mathbf{n} \), where \( \mathbf{t} \) is the boundary tangent vector, (12) induces undesired tangential damping, acceleration, or no-slip behavior (Figure 8, bottom half).

We propose a novel additive ramping procedure that instead enforces \( \psi_g \) by adding a compensating offset. The required offset is determined by evaluating the existing \( \psi \) at the closest boundary...
is the interpolated grid-based $\psi$ value adjustment. Left: Particles with speed-dependent trail lengths. Right: Velocity magnitudes. Top half: Our additive ramping exactly satisfies the boundary with no apparent damping of free-slip velocities. Bottom half: Despite modifying Bridson’s multiplicative ramp to target the correct $\psi_g$ value, tangential damping occurs along the bottom wall and near the solid, since $\partial \psi / \partial n$ is damaged. Note the more pronounced banding in the bottom magnitude plot.

This expression ramps the full correction precisely on at the boundary $d(x)/d_0 = 0$ and blends it smoothly off at the edge of the influence region $d(x)/d_0 = 1$. The inset shows an example input $\psi$ curve (black), and the result after using multiplicative (red) and our additive (green) ramping to $\psi_g = 0$.

Both methods correct the value, but additive ramping better preserves the derivative near the boundary and thus yields more faithful free-slip flow (Figure 8, top half). We again use the analytical curl to determine this added velocity contribution.

By its nature, applying ramping on piecewise flat (polygonal or polyhedral) geometry can introduce some discontinuity in velocity near sharp edges or vertices. This is because the recovered velocity $u^r = \nabla \psi^r$ involves a $\nabla \alpha$ term from (13) (i.e., gradient of the normalized distance field), and $\nabla \alpha$ is not continuous in general. Nevertheless, our Curl-Flow method produces precisely boundary-respecting and divergence-free flows, and the discontinuities are often acceptable in practice, since the applied corrections only occur within one cell-width of obstacles.

In the context of our overall interpolation scheme, the ambient $\psi$ is the interpolated grid-based $\psi$ field and the target boundary $\psi_g$ values are known at solid surface nodes, since they are recovered during sweeping. We apply our ramping strategy to closed exterior domain boundaries and static solid surfaces. In those cases, $\psi_g$ is a constant value, since the boundary’s normal flux is zero. For exterior domain boundaries that have nonzero inflow/outflow, the discrete $\psi$ edge endpoint values of a segment differ (e.g., $\psi_{e_0}, \psi_{e_1}$). One can set the ramping target value $\psi_0$ to the linearly interpolated value of $\psi_{e_0}$ and $\psi_{e_1}$ at the closest point to achieve the desired constant flow in the normal direction across that edge. However, for simplicity, we did not apply our ramping-based enhancement to the exterior domain boundaries with inflow/outflow conditions, since exact enforcement there is not visually critical.

Figure 9 compares direct (bilinear) velocity interpolation vs. our full Curl-Flow method (including additive ramping boundary correction) on a simulated steady-state flow past a disk. To approximate free-slip flow for the comparison direct method, we extrapolate fluid velocities (with solid-normal components projected out) to samples inside the solid [Houston et al. 2003; Rasmussen et al. 2004].

5 CURL-FLOW INTERPOLATION IN 3D

In three dimensions, the scalar stream function $\psi$ is replaced by the vector potential $\Psi = (\psi_x, \psi_y, \psi_z)$. Each velocity component is now dictated by the interactions of two $\psi$ components’ derivatives (e.g., $u = \partial \psi_z / \partial y - \partial \psi_y / \partial z$), which complicates boundary enforcement, especially for irregular geometry. The curl operator also possesses a multi-dimensional null space, necessitating enforcement of a gauge condition to find an appropriate unique $\Psi$.

5.1 Uniform Grids in 3D

We place vector potential components on cell edges and velocity normal components on cell faces [Ando et al. 2015; Bao et al. 2017; Elcott et al. 2007] (Figure 10). Given discretely divergence-free face velocities, we seek corresponding edge vector potential values.

For simplicity, we assume for now that the input axis-aligned exterior domain boundaries have zero normal velocity. We can safely
achieve this with a constant value (i.e., 0) for the tangential components of the boundary \( \psi \) (denoted \( \psi_{\text{tan}} \)). (Our method extends straightforwardly to prescribed inflow/outflow velocities, with appropriate variations in \( \psi_{\text{tan}} \).)

With this choice, the continuous problem we are solving is

\[
\nabla \times \psi = u \quad \text{in} \, \Omega, \\
\psi_{\text{tan}} = 0 \quad \text{on} \, \partial \Omega. \\
\tag{14}
\]

The flux across a given surface, in terms of the vector potential, is

\[
\iint_{S} u \cdot dS = \int_{S} (\nabla \times \psi) \cdot dS = \int_{C} \psi \cdot dr, \\
\tag{15}
\]

where \( S \) is an oriented smooth surface, \( C \) is the oriented boundary curve of the surface, and \( r \) is the boundary tangent direction. (The second equality holds by Stokes’ theorem.) Discretizing (15) with midpoint quadrature for the single face in Figure 10, right, we have

\[
u_{f} h^{2} = \sum_{e \in E} \psi_{e} \cdot (he_{e}) = h(\psi_{y_{0}} + \psi_{z_{1}} - \psi_{y_{1}} - \psi_{z_{0}}),
\tag{16}
\]

where \( h \) is a cell width and \( e_{e} \) is the oriented unit vector along an edge. (Finite differences on (14) yields the same.) Equation 16 relates four unknown edge \( \psi_{e} \) values to one face \( u_{f} \) velocity. Stacking the equations for all grid faces yields a global sparse linear system.

Equation 14 is an inverse curl problem with infinitely many solutions, e.g., a single cell has a seven-dimensional null space: 12 edge \( \psi_{e} \) and five linearly independent face \( u_{f} \) (the sixth is redundant by incompressibility). We select a unique solution using the popular Coulomb gauge condition (with our boundary condition), which enforces \( \nabla \cdot \psi = 0 \). This gauge offers maximal smoothness of the \( \psi \) field, which will be attractive for interpolation.

Directly taking the curl of (14) and assuming \( \nabla \cdot \psi = 0 \) yields

\[
\nabla \times (\nabla \times \psi) = -\nabla^{2} \psi = \nabla \times u, \\
\tag{17}
\]

i.e., a vector Poisson problem for \( \psi \), around three times larger than the pressure projection (e.g., [Ando et al. 2015]). Fortunately, unlike Ando et al., our input discrete velocities are already incompressible, enabling us to solve (14) more efficiently. Observe that

\[
u = \nabla \times \psi = \nabla \times (\psi + \nabla \phi) = \nabla \times \psi',
\tag{18}
\]

where \( \phi \) is an arbitrary scalar field. Since \( \nabla \times \nabla \phi \) is always zero (a vector calculus identity), defining \( \psi' = \psi + \nabla \phi \) gives another valid vector potential field for the same velocity. Leveraging this characterization of the null space, we propose a two step approach. First, use an efficient 3D extension of our 2D parallel sweeping scheme (Section 5.1.1) to find a \( \psi \) field that is velocity-consistent but has arbitrary gauge. Second, modify this \( \psi \) field to satisfy the Coulomb gauge \( \nabla \cdot \psi = 0 \) and our boundary condition \( \psi_{\text{tan}} = 0 \) using a carefully constructed scalar potential \( \phi \) (Section 5.1.2).

5.1.1 Recovering a Vector Potential by Parallel Sweeping. Our first task is to efficiently find a velocity-consistent discrete vector potential field on a box-shaped domain, irrespective of boundary conditions or gauge choice. Our proposed fast 3D parallel sweeping strategy is illustrated in Figure 11.

1. Set \( \psi_{x} = 0 \) (or a constant) everywhere in the domain (Figure 11, top-left). This is safe because the remaining two components \( (\psi_{x}, \psi_{y}) \) still suffice to represent any set of three velocity components \( (u, v, w) \) [Ravu et al. 2016].

2. Compute velocity-satisfying vector potential values for the \( z = z_{\text{min}} \) boundary plane. For zero boundary normal fluxes, we can simply set all \( \psi_{x} \) and \( \psi_{y} \) to zero (Figure 11, top-right).

3. Compute the remaining vector potential values by parallel sweeping. Equation 16 and \( \psi_{z} = 0 \) give \( \psi_{y_{0}} = \psi_{y_{1}} - hu \) (Figure 11, bottom-left), and likewise \( \psi_{z_{1}} = \psi_{z_{0}} + hu \) (Figure 11, bottom-right). In like manner, values of \( \psi_{x_{m}} \) and \( \psi_{y_{n}} \), along with the discrete velocities, dictate \( \psi_{x_{m+1}} \) and \( \psi_{y_{n+1}} \) as we sweep across the entire domain.

When this process ends, the velocity condition (16) is met on all grid faces, \( \psi_{z} = 0 \) everywhere, and we have zero \( \psi_{x} \) and \( \psi_{y} \) values on all the outer boundaries except for \( z = z_{\text{max}} \). (For nonzero boundary fluxes, more \( \psi_{x} \) and \( \psi_{y} \) values will be nonzero.) We must next modify this \( \psi \) to enforce the desired gauge and boundary conditions.

Fig. 10. Discretization in 3D: Discrete vector potential samples are located on cell edges (circles) and velocity samples are on cell faces (squares). Red, green, and blue represent \( x \), \( y \), and \( z \) components, respectively.

Fig. 11. Parallel Sweeping in 3D: Top-left: Set all \( \psi_{z} \) values to be zero. Top-right: Compute satisfying \( \psi_{x} \) and \( \psi_{y} \) values at \( z = z_{\text{min}} \). In this example they are all set to zero. Bottom-left: Starting from \( \psi_{y} \) values at \( z = z_{\text{min}} \), we can compute all \( \psi_{y} \) values in the entire domain in one sweep using the relation \( \psi_{y_{0}} = \psi_{y_{1}} - hu \). Bottom-right: Starting from \( \psi_{x} \) values at \( z = z_{\text{min}} \), we can compute all \( \psi_{x} \) values in the entire domain in one sweep using the relation \( \psi_{x_{1}} = \psi_{x_{0}} + hu \).
The desired boundary condition

\[ \phi \text{ on } \partial \Omega. \]  

Given \( \psi_{\text{tan}} \), already known from parallel sweeping, we must find \( \phi_{\text{BC}} \) on boundary nodes. Furthermore, since our sweeping process ensured that the only nonzero boundary \( \psi_{\text{tan}} \) values left to be eliminated are \( \psi_x \) and \( \psi_y \) on the \( z = z_{\text{max}} \) plane, finding nonzero \( \phi_{\text{BC}} \) values only on \( z = z_{\text{max}} \) suffices. We set the outer boundary loop of nodal \( \phi_{\text{BC}} \) values on this plane (black/white diamonds in Figure 12) to zero, and compute the interior \( \phi_{\text{BC}} \) values in the plane by traversing the interior edges using \( \phi_{\text{next}} = \phi_{\text{prev}} - h \psi_x \).

Adding \( \nabla \phi_{\text{BC}} \) directly to \( \phi \) (including any interior edges touching the nonzero \( \phi \) values) would satisfy our exterior boundary condition without breaking consistency between the discrete \( u \) and \( \phi \). However, the (arbitrary) gauge of the resulting interior field still offers no guarantees on the presence or absence of large, discontinuous jumps [Silberman et al. 2019]. Thus, while applying interpolation and the analytical curl will yield some pointwise divergence-free velocity field, its observed pointwise behavior can be quite irregular depending on interactions between the discrete data jumps and componentwise interpolants, because velocity is determined by differences of vector potential component derivatives. Fortunately, we can enforce the Coulomb gauge on the interior to achieve an optimally smooth vector potential field for interpolation.

Defining a new global nodal \( \phi \) field, we apply a gauge correction similar to Silberman et al. [2019] (although they consider a Fourier-based solution with exterior boundary conditions at infinity and do not handle interior solids). The Coulomb condition says that \( \nabla \cdot \psi' = \nabla \cdot (\phi + \nabla \phi) = 0 \), giving a node-based scalar Poisson problem for \( \phi \) with Dirichlet boundary conditions:

\[
\nabla \cdot \nabla \phi = -\nabla \cdot \psi \quad \text{in } \Omega,
\]

\[
\phi = \phi_{\text{BC}} \quad \text{on } \partial \Omega.
\]

For the box-shaped domains we consider, this Dirichlet problem is amenable to solution with a classic fast Poisson solver [Van Loan 1992] based on the Discrete Sine Transform (DST), even after we incorporate irregular obstacles (Section 5.2). After solving for \( \phi \) we update the vector potential as \( \psi' = \psi + \nabla \phi \).

5.1.3 Inflow/outflow exterior boundaries. The extension to exterior domain boundaries with nonzero fluxes (inflow and outflow) can be carried out similarly to the 2D case. For example, to match prescribed \( u \) velocities on the \( x \) boundary, we can use linear \( \psi_x \) values in the \( y \) direction on the boundary, while keeping \( \psi_y \) values at zero. The sweeping and gauge correction steps are easily modified to preserve these boundary values throughout.

5.1.4 Interpolation. Similar to 2D, we handle 3D uniform grid interpolation using low-order dyadic spline kernels, separately on each staggered component of \( \psi \), but choose their polynomial degrees to ensure continuous velocities as follows. The curl operator applied to one component of \( \psi \) (e.g., \( \psi_x \)) involves its partial derivatives in the other two axes (e.g., \( \partial \psi_x / \partial y \) and \( \partial \psi_x / \partial z \)). To ensure the resulting velocity is at least (piecewise) linear (and continuous) in all directions, we use a mix of linear and quadratic kernels (e.g., for \( \psi_x \), linear \( x \), quadratic \( y \) and \( z \)). Uniformly quadratic or higher order interpolants would also suffice, in exchange for higher cost.

5.2 Cut-Cells in 3D

The interference of cut-cell solids requires adaptations to our parallel sweeping and gauge correction steps, and a modified 3D ramping strategy to exactly enforce the desired boundary behavior.

5.2.1 Parallel Sweeping with 3D Cut-Cells. By carefully removing redundant DOFs and choosing traversal orders in Sec. 5.1.1, we efficiently obtained velocity-consistent \( \phi \) values on uniform grids. With cut-cells present, the number and orientation of vector potential components on some faces change, and thus our sweeping approach must be adapted. Conceptually, we achieve this by converting the
where we collapse them and rescale the fluid edges. Variables, we can finally rewrite this in a form consistent with the vector potential (the fluid parts of the edge are unchanged). Then, area. Letting \( \psi \) edge and face fixed, but rescale potentials by length and velocity by uniform case of (16). We hold the weighted contribution of each \( \text{F}, \) partial fluid edges, is the area of a regular (non-cut) cell face, the \( l \) face in Figure 14(a), left, using (15) as

Second, we assume that solid portions of partially cut grid-edges to the solid have zero normal flux (purple triangle in Figure 14(a)).

To achieve this mathematically, we rely on two assumptions. First, these two (sub-)faces imply two equations,

\[
\begin{align*}
\psi_{x_0} & = \frac{l_0}{l} \psi_{x_1}, \quad \psi_{z_1} = \frac{h}{h} \psi_{z_0}, \\
\psi_{x_0} & = \psi_{x_1}, \\
\psi_{z_1} & = \psi_{y_1}, \\
0 & = l_2 \psi_{x_1} - h \psi_{y_1},
\end{align*}
\]

where in the second equation we again assume zero potential on the two axis-aligned purple solid edges. Rescaling and sweeping suffices to recover the potentials, as before.

5.2.2 Approximate Gauge Correction with 3D Cut-Cells. Applying gauge correction is more difficult with polyhedral cut-cell solids, since discrete vector potential edge components can correspond to arbitrary, rather than Cartesian, directions. A polyhedral PDE solver could be used, e.g., mimetic finite differences [Lipnikov et al. 2006] or the diamond Laplacian [Bunge et al. 2021]. However, such a solver is more costly and interpolating the resulting non-Cartesian edge components would also be unwieldy. Moreover, as in 2D, we still require a ramping correction to precisely respect solid boundaries. Since our motivation for gauge correction is simply to ensure smoothness, minor local deviations from the Coulomb gauge are acceptable. We therefore enforce the gauge in a simple but approximate manner: we use the rescaled data from the voxelized view of our geometry determined above (Figure 14), and perform gauge correction on the entire uniform grid (including through the interior of solids). This voxelized approach also still lets us retain the use of the fast DST Poisson solver. Thus, our method efficiently provides smooth axis-aligned \( \psi \) values everywhere, which we use for B-spline kernel interpolation as in the uniform case.

5.3 Precise boundary enforcement in 3D
As in 2D, we propose an additive ramping-based approach for exact boundary enforcement in 3D, applied as a correction atop the approximate \( \psi \) values produced by grid interpolation. The 3D adaptation requires caution in treating the interacting \( \psi \) components.

5.3.1 Exterior domain boundaries. The domain exterior consists of axis-aligned planes with two tangential components of \( \psi \) lying on each boundary plane (Figure 15(a)). Recall that for a \( u \)-face, the flux is determined by \( \psi' \) and \( \psi'' \) through \( u = \frac{\partial \psi'}{\partial n} - \frac{\partial \psi''}{\partial n} \). To achieve exactly zero flux everywhere on the plane and avoid leakage, these terms must precisely cancel: we set the discrete \( \psi' \) and \( \psi'' \) values to zero for simplicity, and consequently ramp the pointwise \( \psi' \) and \( \psi'' \) values to zero as a particle approaches the border. (A careless choice of the ramp values can badly distort the velocity, as we saw in 2D in Figure 7.) Since we apply ramping to each component of \( \psi \) independently, this ramping strategy is identical to the 2D case (Section 4.2.3). As in 2D, we do not enforce exact ramping on inflow/outflow boundaries as its effects are not visually apparent.

5.3.2 Solid obstacles. For our explicit representation of the level set-based solid obstacles, we use triangle meshes obtained from
Fig. 15. **Boundary enforcement:** The black and red disks represent a particle near the boundary and its closest point on the boundary, respectively. (a) For planar exterior boundaries, the discrete \( \psi \) values (e.g., \( \psi_{y0}, \psi_{y1}, \psi_{y2} \)) are zero by our boundary conditions, and the relevant axis components \( (\psi_y(x) \text{ and } \psi_x(x)) \) are ramped to zero for exact enforcement. (b) For triangulated solids, we first find discrete \( \psi \) values at the triangle vertices (e.g., \( \psi_{y0}, \psi_{y1}, \psi_{y2} \)) which imply a perfectly tangential surface velocity, under barycentric interpolation of nodal \( \psi \). When querying velocities, we ramp the full \( \psi \) vector towards this interpolated surface \( \psi \) at the closest point.

marching cubes. A key component of exact boundary enforcement with ramping is the proper choice of target \( \psi \) values (i.e., \( \psi_{y0} \)) on the solid surface. Specifically, \( \psi_{y0} \) should produce a zero normal velocity,

\[
\mathbf{u} \cdot \mathbf{n} = (\nabla \times \psi_y) \cdot \mathbf{n} = 0, \tag{24}
\]

where \( \mathbf{n} \) represents the solid normal vector.

For exterior boundaries, we were able to simply set constant discrete \( \psi_{y0} \) values in the solve, and by using those same constant values for the target \( \psi_y \) during ramping, (24) was trivially satisfied. However, one cannot in general find a single constant vector \( \psi_y \) that precisely compensates the existing (spatially varying) ambient interpolated \( \psi \) on irregular solids. Moreover, recall that an arbitrary choice of \( \psi_y \) will still severely distort the nearby tangential velocities, even if it satisfies (24).

We instead leverage the characteristics of triangular barycentric coordinates: the gradient of the barycentric interpolant is constant, thus \( \nabla \times \psi_y \) yields a constant velocity (per triangle). (We also tried other interpolants satisfying (24) continuously on a triangulated surface (piecewise constant, Whitney), but found that barycentric produced the smoothest velocity fields.) Therefore, an appropriate choice of \( \psi \) at the triangle vertices \( (\psi_{y0}, \psi_{y1}, \psi_{y2} \text{ in Figure 15(b)}) \) can satisfy (24), exactly and continuously. At the same time, we wish to minimize the perturbation of the velocity field induced by ramping, which we can do by encouraging the discrete \( \psi_y \) values at triangle vertices to be as close to the ambient \( \psi \) (i.e., the interpolated pointwise \( \psi \) before applying ramping) as possible. This yields an equality constrained quadratic minimization problem:

\[
\begin{align*}
\arg\min_{\mathbf{y}} & \quad \frac{1}{2}(\mathbf{y} - \mathbf{\psi}_{\text{amb}})^T(\mathbf{y} - \mathbf{\psi}_{\text{amb}}) \\
\text{subject to} & \quad (\nabla \times \sum_{i \in \{0,1,2\}} w_i(x)\mathbf{y}_t) \cdot \mathbf{n}_t = 0 \quad \text{for } t \in T,
\end{align*}
\]

where \( T \) is a set of (connected) solid triangles, \( w_i(x) \) is the barycentric coordinate function for the \( i^{th} \) vertex for a spatial position \( x \), \( t_i \) is the global vertex index of the \( i^{th} \) vertex in the \( i^{th} \) triangle, \( \mathbf{n}_t \) is the triangle normal, and \( \mathbf{\psi}_{\text{amb}} \) is a stack of the ambient \( \psi \) values at the triangle vertices (before ramping). If we write the constraints as \( A\mathbf{y} = 0 \), the optimality conditions yield the linear system

\[
\begin{bmatrix}
I & A^T \\
A & 0
\end{bmatrix} \begin{bmatrix}
\mathbf{y} \\
\mathbf{\lambda}
\end{bmatrix} = \begin{bmatrix}
\mathbf{\psi}_{\text{amb}} \\
0
\end{bmatrix} \tag{25}
\]

or \( AA^T\mathbf{\lambda} = A\mathbf{\psi}_{\text{amb}} \). We can find the desired discrete \( \psi_y \) at the solid vertices via \( \mathbf{y} = \mathbf{\psi}_{\text{amb}} - A^T\mathbf{\lambda} \). Since the normal component of velocity only depends on the tangential components of \( \psi \), one need only ramp the tangential components using the target values of \( \psi_{y0} = \mathbf{\psi}_y - (\mathbf{\psi}_y \cdot \mathbf{n})\mathbf{n} \). For ramping, \( \psi_y \) and \( \psi_{y0} \) can be queried by barycentric interpolation \( y \) in the closest triangle.
Letting $\psi_{nor}(x)$ be the normal component of the ambient $\psi$, the final modified pointwise $\psi$ with ramping is

$$\psi'(x) = \psi_{tan}(x) + \psi_{nor}(x)$$

$$= \psi_{tan}(x) + (\alpha(x, cp(x)) - 1)(\psi_{tan}(cp(x)) - \psi_{g,tan}(cp(x))) + \psi_{nor}(x)$$

(26)

Again, we use $cp(x)$ to represent the closest point on the solid triangles and $\alpha(x) = \text{ramp}(d(x/d_0))$ for a smooth transition toward $\psi_{g,tan}$. Here, $d_0$ is the influence radius ($b$ in our case), and $d()$ is the distance to the closest point $cp(x)$. We again employ the smoothstep function for $\text{ramp}()$. This approach is much like (13) in 2D, except that the target $\psi_{g,tan}$ is not a constant since it is computed from barycentric interpolation.

The corresponding free-slip velocity is

$$u'(x) = \nabla \times (\psi_{tan}(x) + \psi_{nor}(x))$$

$$= \nabla \times \psi(x) + \nabla \alpha(x, cp(x)) \times (\psi_{tan}(cp(x)) - \psi_{g,tan}(cp(x))) + (\alpha(x, cp(x)) - 1)(\nabla \times (\psi_{tan}(cp(x)) - \psi_{g,tan}(cp(x))))$$

(27)

As the position $x$ approaches $cp(x), \nabla \times \psi_{tan}(x)$ and $\nabla \times \psi_{tan}(cp(x))$ cancel each other out, and due to barycentric interpolation, $\nabla \times \psi_{g,tan}(cp(x))$ is constant per triangle, which we constrained to have zero normal velocity in (25); thus exact zero flux on the solid surface. (Other components do not influence normal velocity.)

Due to the piecewise constant component from barycentric interpolation in (27), it is possible for additional velocity discontinuities to be introduced, beyond those coming from non-smooth changes in $cp(x)$ (Section 4.2.3). However, we found these effects to be a reasonable tradeoff for incompressibility and precise boundary satisfaction. Moreover, the correction applied by our scheme is often small and any possible discontinuities can only occur within one cell width of solid obstacles or domain boundaries, whereas the discontinuities mentioned in related work (e.g., DG methods (Section 2.2.1) and most direct incompressible velocity interpolants (Section 2.2.2)) arise between adjacent elements or cells everywhere throughout the domain, even in the absence of obstacles.

To illustrate the improved behavior of 3D Curl-Flow with cut-cells in a basic scenario, we consider a flow past a solid sphere with a steady state velocity field (Figure 16), and release a single planar layer of passive particles. With trilinear advection, many of the particles collide with the obstacle, leaving a large gap in the particle field. With Curl-Flow, the particles instead flow more naturally around the obstacle.

### 6 RESULTS AND DISCUSSION

To consistently compare our Curl-Flow method against direct velocity interpolation while isolating interpolation-related effects, most of our comparisons change only the interpolation of velocity used for passive particle tracing, keeping the underlying simulated flow data the same. (The exception are tests that apply Curl-Flow to tracing the FLIP particle paths for FLIP advection.) Whenever a particle penetrates a solid object due to poor quality velocity fields, large timesteps, or insufficiently accurate path integration, a typical choice is to “resolve” the collision by projecting the particle back out of the solid, although this exacerbates clumping. To visually highlight the pervasiveness of such errors, we freeze penetrating particles in place, unless otherwise noted (such as in Figure 16(b)).

Simulation timings were gathered on a 2.8 GHz, 4-Core Intel Core i7 processor.

#### 6.1 Particle Distribution Comparisons in 2D

The 2D results of Figures 3, 7, 8, and 9 used static velocity fields for illustration; below, we consider time-evolving (i.e., dynamic) flow simulations.

In Figure 17, we densely seed passive particles with blue noise sampling and advect them under a dynamic simulated 2D vector field with turbulent forcing on a $20 \times 20$ grid. Frames 1, 50, 100, and 150 are shown from left to right. Particle distribution remains more uniform with Curl-Flow.
problematic in that setting. Figure 19 (bottom half) applies FLIP using Curl-Flow for particle tracing (but not particle-to-grid (P2G) or grid-to-particle (G2P) transfers). Aside from FLIP particle seeding at the start and left inflow (8 particles per grid cell), no artificial density interventions were applied, e.g., reseeding, particle position adjustment, etc. Since P2G fails for empty cells, such cells fall back to semi-Lagrangian advection. We observe trailing gaps with direct velocity interpolation and, furthermore, its motion is more damped compared to FLIP with Curl-Flow.

6.2 Particle Distribution Comparisons in 3D

Static Flow In A Box. To explore incompressibility’s effects in 3D at near- and sub-grid scales, Figure 1 compares our full Curl-Flow method (including exterior boundary ramping) against direct velocity interpolants with (componentwise) trilinear and monotonic tricubic schemes [Fedkiw et al. 2001; Fritsch and Carlson 1980], as these are simple and common choices. The data is a static, extremely coarse $10 \times 10 \times 10$ discretely incompressible discrete vector field. The direct interpolants quickly exhibit particle clustering and spreading, forming visible low density regions and dense ring-like structures. By contrast, our result remains more uniformly distributed over time, despite no particle resampling, positional perturbations, or other remedies being applied. Clearly our method succeeds at enforcing incompressibility at even the finest scales.

Dynamic Flows Past Smooth and Jagged Obstacles. At medium scales, we next compare Curl-Flow to direct velocity interpolation under separate 3D dynamic horizontal flows past a sphere and a jagged obstacle (Figure 20). Here we consider grids ranging from $40 \times 20 \times 20$ to $80 \times 60 \times 60$. Only the particles within a narrow slice plane are shown to better visualize their density. The particles are purely passive, but for illustration we seed them throughout the domain at a density representative of typical particle-in-cell schemes, i.e., 8 per cell [Zhu and Bridson 2005]. As in the corresponding 2D examples, Curl-Flow retains more uniform particle sampling.

The average computational costs per timestep for our method applied to the dynamic flows above are given in Table 1. For reference, we also include timings for the underlying fluid simulator’s pressure projection, as it is often a bottleneck in fluid solvers. For these tests, we solve both pressure projection and gauge correction with the basic conjugate gradient method in the Eigen library [Guennebaud et al. 2010], with relative tolerance $1 \times 10^{-8}$. We used Eigen’s LDLT direct solver for the solid surface problem (25), since we found it can be ill-conditioned. The observed total cost of reconstructing $\phi$ (for the grid and solid surfaces) is noticeably less than for pressure projection (around 50% – 75% as much). The cost of advecting particles with our interpolant is slightly more than for trilinear interpolation (around 10% – 30% more).

6.3 Smoke Simulations and Performance

In the following two examples, we consider rising smoke plumes in $150^3$ grids with different obstacles: a stylized calf and Houdini’s rubber toy (Figure 22). For a simple age-based dissipation effect, we removed particles after 4 seconds of existence, but to emphasize collision errors we did not delete any particles that collided with the obstacle. We supplied up to 4M purely passive smoke particles for each example.

Our ramping method enforces a pointwise boundary-respecting velocity field under Curl-Flow, but particles advected with discrete timesteps will still collide with solid obstacles, especially with fast dynamic flows and/or complex solids. For the smoke examples (our version and the baseline), we apply substepping to (only) particle advection to partially ameliorate this innate problem, constraining the timestep so that particles cannot move more than one grid cell per substep. Using more substeps would further reduce collisions, at the cost of increasing computational time. We additionally apply a simple “collision-aware” substepping upon first contact: if a particle hits an obstacle, we push it slightly back from the hit position (10% of the distance the particle moved) and then perform a new partial substep from that position for the remaining time. (If it is still colliding afterwards, it is then frozen.)

To demonstrate Curl-Flow’s practicality, these smoke plume tests were carried out by implementing our method into Houdini’s smoke solver (Pyro), with various modifications (e.g., we used our own cut-cell pressure projection, solved with Eigen’s CG). Unlike the prior examples, we used a MacCormack scheme (Houdini’s default) for velocity advection and we used our DST Poisson solver for fast gauge correction. In both examples, Curl-Flow shows superior boundary-respecting behavior; particles flow over and around the shape, instead of into it and halting. This effect can be observed by inspecting the density and brightness of smoke in the later images. Timings are provided in Table 2.

Curl-Flow incurs preprocessing costs at each step to reconstruct discrete $\phi$ on the regular grid and on solid vertices, i.e., primarily a volumetric Poisson problem and a surface-based least squares problem. Fortunately, our DST solver makes gauge correction efficient and the fact that the surface solve is lower dimensional likewise keeps its scaling reasonable. We investigate these costs in the $\phi$ reconstruction scaling tests of Table 3. At $256^3$, $\phi$ construction is two orders of magnitude faster than cut-cell pressure projection.
Fig. 19. **Dynamic 2D Flow in A Wind Tunnel Past A Solid:** Passive tracer particles (top two rows) and FLIP particles (bottom two rows) undergo a dynamic horizontal flow past static obstacles. Particles colliding into objects are halted in place. Direct velocity interpolation creates spurious gaps (blue), even at triple the grid resolution; Curl-Flow interpolation tightly and incompressibly follows the solid objects, significantly reducing the gaps in the flow (red).

Fig. 20. **Dynamic 3D Flow in A Wind Tunnel Past A Solid (middle frame):** A slice-plane view of particles undergoing a time-dependent horizontal flow past a solid object in 3D. The color of the particles represents the particle densities: the density increases from red to yellow. Particles colliding into objects are halted in place for illustration. Regardless of the grid resolution, direct velocity interpolation creates spurious gaps, while our Curl-Flow interpolation tightly follows the solid object, significantly reducing the gaps in the flow and maintaining better particle distribution.

with Eigen’s CG (Fourier methods do not support cut-cells). With major implementation effort, advanced multigrid and GPU solvers can sharply reduce the cost of pressure projection [Raateland et al. 2022; Shao et al. 2022]; however, even in the worst case, since both are Poisson problems, pressure projection and gauge correction would have around the same cost (e.g., see rows 3 and 4 of Table 4).

The remaining additional overhead of our method arises during particle advection: Curl-Flow has a larger stencil size for interpolation (quadratic vs. linear), and our ramping strategy requires
Table 1. Average computational time per timestep for didactic examples.

| Examples | Interpolation Method | Pressure Projection (s) | Grid \( \psi \) Construction (s) | Solid \( \psi \) Construction (s) | Particle Advection (s) | # Particles | # Solid Vertices (Triangles) |
|----------|----------------------|-------------------------|-------------------------------|---------------------------------|------------------------|-------------|-------------------------------|
| Flow Past A Sphere (40 \( \times \) 20 \( \times \) 20) | Direct Velocity Interpolation | 2.214 \( \times \) 10\(^{-2}\) | – | – | 3.962 \( \times \) 10\(^{-1}\) | 117,482 | 726 |
| | Curl-Flow | 9.040 \( \times \) 10\(^{-3}\) | 5.887 \( \times \) 10\(^{-3}\) | 4.443 \( \times \) 10\(^{-1}\) | 117,483 | (1448) |
| Flow Past A Sphere (80 \( \times \) 40 \( \times \) 40) | Direct Velocity Interpolation | 4.268 \( \times \) 10\(^{-1}\) | – | – | 3.010 | 949,272 | 2934 |
| | Curl-Flow | 1.812 \( \times \) 10\(^{-1}\) | 3.888 \( \times \) 10\(^{-2}\) | 3.708 | 949,277 | (5864) |
| Flow Past A Jagged Obstacle (40 \( \times \) 30 \( \times \) 30) | Direct Velocity Interpolation | 9.942 \( \times \) 10\(^{-2}\) | – | – | 8.158 \( \times \) 10\(^{-1}\) | 271,144 | 3140 |
| | Curl-Flow | 4.262 \( \times \) 10\(^{-2}\) | 2.907 \( \times \) 10\(^{-2}\) | 1.050 | 271,148 | (6272) |
| Flow Past A Jagged Obstacle (80 \( \times \) 60 \( \times \) 60) | Direct Velocity Interpolation | 2.070 | – | – | 6.885 | 2,035,118 | 12454 |
| | Curl-Flow | 9.001 \( \times \) 10\(^{-1}\) | 2.070 \( \times \) 10\(^{-1}\) | 7.723 | 2,035,124 | (24900) |

Fig. 21. A Smoke Plume Simulation with A Solid Shaped like A Calf. In the direct velocity interpolation case, a large number of particles erroneously collide with the solid and are then frozen, especially around the ears and underside of the head; with Curl-Flow, more particles flow smoothly over and around the object’s surface and since more smoke remains active it is denser and brighter (middle column, above the face, and see video).

additional operations, e.g., finding the closest point on the solid, evaluating a second interpolation, etc., unlike simple grid-based velocity interpolation. In the calf and dragon examples, collided (frozen) particles also no longer undergo advection, which significantly benefits the direct approach. In Figure 21, 1,508,617 (top) and 885 (bottom) particles are frozen and in Figure 22, 2,136,520 (top) and 2,236 (bottom) particles are frozen at the last frame. Advection costs are heavily dependent on the number of active particles and can become the bottleneck for massive particle counts. We anticipate that interpolation could be further optimized. Most importantly, since particle advection is done in a fully parallel manner, more compute cores immediately translate into speedups. Genuinely passive particles can also be traced in an entirely parallel post-process, after simulation completes.
Fig. 22. **A Smoke Plume Simulation with A Rubber Toy.** The lingering outline of dense smoke from collided particles (frozen for emphasis) is conspicuous on the belly in the direct velocity interpolation case. For Curl-Flow, the smoke instead flows around and past the object, so it remains visibly denser after going by (middle and right columns).

Table 2. Average computational time per timestep for smoke simulations of Figure 21 and Figure 22. 400K particles were supplied every second, and they fade away after 4 seconds.

| Interpolation Method | Pressure Projection (s) | Grid $\varphi$ Construction (s) | Solid $\varphi$ Construction (s) | Particle Advection (s) | # Solid Vertices (Triangles) |
|----------------------|-------------------------|---------------------------------|---------------------------------|-----------------------|-----------------------------|
| Stylized Calf (150³) | 3.898×10⁻¹ | – | 1.213 | 16,450 (32,896) |
| Curl-Flow (150³)    | 3.807×10⁻¹ | 3.571×10⁻¹ | 2.749×10¹ |
| Stylized Calf (32³) | 4.875×10⁻² | 6.540×10⁻³ | 4.026×10⁻³ | 1,126 (2,252) |
| Stylized Calf (64³) | 8.419×10⁻¹ | 2.920×10⁻² | 2.483×10⁻² | 2,974 (5,948) |
| Stylized Calf (128³) | 1.984×10¹ | 2.158×10⁻¹ | 1.987×10⁻¹ | 11,946 (23,888) |
| Stylized Calf (256³) | 3.685×10² | 1.713 | 1.570 | 47,930 (95,856) |
| Rubber Toy (32³) | 4.040×10⁻² | 1.129×10⁻² | 1.849×10⁻³ | 1,836 (3,664) |
| Rubber Toy (64³) | 8.998×10⁻¹ | 2.954×10⁻² | 3.257×10⁻² | 7,508 (15,008) |
| Rubber Toy (128³) | 1.979×10¹ | 2.111×10⁻¹ | 2.509×10⁻¹ | 30,600 (61,192) |
| Rubber Toy (256³) | 3.844×10² | 1.705 | 2.166 | 123,284 (246,560) |

Table 3. Average computational time per timestep for discrete field construction of Curl-Flow at various scales.

| Pressure Projection (s) | Grid $\varphi$ Construction (s) | Solid $\varphi$ Construction (s) | # Solid Vertices (Triangles) |
|-------------------------|---------------------------------|---------------------------------|-----------------------------|
| Stylized Calf (32³) | 4.875×10⁻² | 6.540×10⁻³ | 726 (1,452) |
| Stylized Calf (64³) | 8.419×10⁻¹ | 2.920×10⁻² | 2,974 (5,948) |
| Stylized Calf (128³) | 1.984×10¹ | 2.158×10⁻¹ | 11,946 (23,888) |
| Stylized Calf (256³) | 3.685×10² | 1.713 | 47,930 (95,856) |
| Rubber Toy (32³) | 4.040×10⁻² | 1.129×10⁻² | 1,836 (3,664) |
| Rubber Toy (64³) | 8.998×10⁻¹ | 2.954×10⁻² | 7,508 (15,008) |
| Rubber Toy (128³) | 1.979×10¹ | 2.111×10⁻¹ | 30,600 (61,192) |
| Rubber Toy (256³) | 3.844×10² | 1.705 | 123,284 (246,560) |

6.4 Vector Potential Reconstruction Comparisons

Several alternative, but more costly, methods to reconstruct edge-based vector potential fields from face-based vector fields have previously been proposed [Ando et al. 2015; Bao et al. 2017; Sato et al. 2015; Silberman et al. 2019]. Since those methods mostly do not handle cut-cell obstacles, to fairly compare computational costs we consider a flow in an empty box domain. We adapted the methods of Ando et al. and Silberman et al. into our framework, replacing our $\varphi$ reconstruction step. We consider the combined cost of pressure projection and $\varphi$ reconstruction for each method in Table 4, since Ando et al. achieve both simultaneously. For a fair comparison, the first four rows use Eigen’s CG for all solves. Since our DST-based gauge correction can always be used whether obstacles are present or not, we present this result in its own row. Despite different
Table 4. Average computational time per timestep for discrete $\mathbf{\varphi}$ construction on the $150^3$ grid simulation of Figure 23 for various methods.

| Method                                                                 | Initial $\mathbf{\varphi}$ Construction (s) | Gauge Correction (s) | Total $\mathbf{\varphi}$ Construction (s) |
|-----------------------------------------------------------------------|---------------------------------------------|----------------------|--------------------------------------------|
| Vector Poisson, Dimensionally-Coupled (Ando et al. [2015])           | -                                           | -                    | 152.0                                      |
| Vector Poisson, Componentwise (e.g., Bao et al. [2017])               | -                                           | -                    | 81.87                                      |
| Pressure Projection + Gauge Correction (CG) (Silberman et al. [2019]) | 37.13                                       | 5.723                | 68.37                                      |
| Ours: Pressure Projection + Gauge Correction (CG) + Parallel Sweeping | 37.13                                       | 0.05904              | 61.51                                      |
| Ours: Pressure Projection + Gauge Correction (DST) + Parallel Sweeping| 37.13                                       | 0.05781              | 37.52                                      |

Comparison to Ando et al. [2015]: Ando et al. solve a single, dimensionally coupled vector Poisson problem for $\mathbf{\varphi}$ from a divergent velocity field, avoiding a separate pressure projection. We modified this approach to satisfy our domain boundary conditions (Section 5.3.1). As shown in Table 4, its computational cost (topmost row) is expensive compared to our method (bottom two rows): about 2.5 – 4× slower for the $150^3$ grid of Figure 23. Unless boundary conditions couple them (as is often the case for free surfaces and some solid boundary treatments [Ando et al. 2015]), the dimensionally-coupled vector Poisson equation can be split into three independent scalar Poisson equations for efficiency. Applying this special case optimization nearly halves the cost (Table 4, second row), but it remains about 1.3× – 2.2× the cost of our approach.

Comparison to Bao et al. [2017]: Bao et al. use a vector potential in their immersed boundary method. They assume divergence-free input and exploit the lack of (cut-cell) obstacles to use the componentwise vector Poisson form (three scalar Poisson problems). They assume periodic boundaries to solve with the FFT. For a fair comparison, we used their (MATLAB) code and replaced only the discrete $\mathbf{\varphi}$ construction step with our method, similarly using FFT for gauge correction. Our method for $\mathbf{\varphi}$ construction is $\sim$2.5× faster, as shown in Table 5.

Although our focus is fluid velocity interpolation, vector potential reconstruction has already been used in graphics (for fluid control [Sato et al. 2021, 2015] and visualization [Biswas et al. 2016]), as well as electromagnetics and beyond. Our new approach can thus offer immediate speedups in all of these domains, e.g., based on Table 4.
6.5 Convergence of Curl-Flow Velocity Interpolation

Since Curl-Flow uses intermediate vector potentials rather than directly interpolating grid velocities, one may ask if it is a valid interpolant at all: does the resulting continuous field converge? We study this question by evaluating how well the interpolated field agrees with an input analytically divergence-free field under refinement. Since the method’s input should be a discretely divergence-free velocity field from an input analytical field and apply pressure projection for discrete incompressibility, then apply the chosen interpolant (at a fine sampling of points) and evaluate their error with respect to the original analytical field. We use the field

\[ u(x, y) = (\sin(2\pi x)\cos(2\pi y), -\cos(2\pi x)\sin(2\pi y)), \]

shown in Figure 25(a). Since Curl-Flow’s quadratic kernel will query some sample positions outside the domain, we linearly extrapolate \( \psi \) values to such samples for the convergence test. Direct (bilinear) velocity interpolation yields second-order convergence and Curl-Flow shows similar behavior (Figure 25).

7 CONCLUSIONS AND FUTURE WORK

Large time steps in particle tracing, poorly enforced boundaries, and inadequate pressure solver tolerances are known to cause density/volume drift; many post-compensation strategies have been developed [Ando et al. 2012; Kugelstadt et al. 2019; Sato et al. 2018b; Takahashi and Lin 2019]. Our work is the first in computer graphics to identify divergent velocity interpolation as an additional factor. Curl-Flow, tailored to plug into popular grid-based cut-cell fluid simulators, addresses this issue up front by constructing pointwise divergence-free velocity fields that also respect obstacles. It offers better long-term particle distributions and natural motions around obstacles and, when used for FLIP particles, yields more dynamic motion and/or reduces the need for post-compensation. We believe our work opens a new direction in the design of advection schemes for fluid animation and raises compelling questions for future study.

We did not consider moving obstacles, which are a useful element of many practical scenarios. Allowing for non-zero boundary fluxes during vector potential reconstruction seems feasible; however, ensuring exactly collision-safe trajectories near Eulerian cut-cell solids (i.e., the aim of the present work) is significantly complicated by the Lagrangian nature of moving obstacles. We have only considered grid domains that are fully open, fully closed, or “wind-tunnel”-like. More general boundary conditions are an obvious next step, including free surface boundaries for liquid animation.

For our ramping \( \psi \) towards solids, we found that seeking a minimal perturbation to \( \psi \) gave visually natural results, but other criteria are possible (e.g., controlling \( \nabla \times \psi \)). A drawback of boundary handling in Curl-Noise-type schemes is that, though always incompressible, they can yield discontinuities when the closest polygon facet changes. A challenging extension would be to develop a strictly continuous variant of Curl-Noise, perhaps by incorporating a smooth distance measure [Madan and Levin 2022].

While not a salient issue for simply tracing particle paths, interpolation of higher order than linear can yield overshooting. We sidestepped this possibility in our FLIP tests by not using the Curl-Flow field for P2G or G2P transfers: any new extremum will not feed back into the dynamic velocity field. Using Curl-Flow for G2P (or all the interpolations within a semi-Lagrangian update) may necessitate some form of limiter. Straightforward velocity clamping [Selle et al. 2008] would sacrifice incompressibility, so falling back to linear Curl-Flow (with kinks) may be preferable.

Lastly, the ability to edit fluid simulations in an efficient post-process remains highly desirable [Pan et al. 2013; Sato et al. 2018a] due to their cost and limited controllability. Our work, and that of Sato et al. [2021; 2015], hint that vector potentials may unlock fast, divergence-free editing capabilities by combining discrete simulation tools and continuous procedural tools.

REFERENCES

R. Albanese and G. Rubinacci. 1990. Magnetostatic field computations in terms of two-component vector potentials. Internat. J. Numer. Methods Engrg. 29, 3 (mar 1990), 515–532. https://doi.org/10.1002/nme.1620290305

Ryoichi Ando, Nils Thuerey, and Chris Wojtan. 2015. A stream function solver for liquid simulations. ACM Transactions on Graphics 34, 4 (Jul 2015), 53:1–53:9. https://doi.org/10.1145/2766925.2767973

Ryoichi Ando, Nils Thuerey, and Reiji Tsuruno. 2012. Preserving fluid sheets with adaptively sampled anisotropic particles. IEEE Transactions on Visualization and Computer Graphics 18, 8 (2012), 1202–1214.

ACM Trans. Graph., Vol. 41, No. 6, Article 243. Publication date: December 2022.
Alexis Angelides and Fabrice Neyret. 2005. Simulation of smoke based on vortex filament primitives. In Proceedings of the 2005 ACM SIGGRAPH/Europographics symposium on Computer animation. ACM, 87–96.

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.

Dinhaw S Balsara. 2001. Divergence-free adaptive mesh refinement for magnetohydrodynamics. J. Comput. Phys. 174, 2 (2001), 614–648.

Dinhaw S Balsara. 2004. Second-order-accurate schemes for magnetohydrodynamics with divergence-free reconnection. The Astrophysical Journal Supplement Series 151, 1 (2004), 149.

Dinhaw S Balsara. 2009. Divergence-free reconstruction of magnetic fields and WENO schemes for magnetohydrodynamics. J. Comput. Phys. 228, 14 (2009), 5040–5056.

Yuanxun Bao, Aleksandar Donvec, Boyce E Griffith, David M McQueen, and Charles S Peskin. 2017. An Immersed Boundary method with divergence-free velocity interpolation and force spreading. Journal of computational physics 347 (2017), 183–206.

Ayan Biswas, Richard Streitl, Jonathan Woodring, Chun-Ming Chen, and Han-Wei Shen. 2016. A scalable streamline generation algorithm via flux-based isosurfacet extraction. In Proceedings of the 16th Eurographics Symposium on Parallel Graphics and Visualization. 69–78.

R. Bridson. 2015. Fluid Simulation for Computer Graphics, Second Edition. Taylor & Francis.

Robert Bridson, Jim Haoziam, and Marcus Nordenstam. 2007. Curl-Noise for Procedural Fluid Flow. ACM Trans. Graph. 26, 3 (July 2007), 46–es. https://doi.org/10.1145/1275777.1276435

Tyson Brochu, Todd Keeler, and Robert Bridson. 2012. Linear-time smoke animation with vortex sheet meshes. In Proceedings of the ACM SIGGRAPH/Europographics Symposium on Computer Animation. Europographics Association, 87–95.

Astrid Bunge, Maria Botsch, and Marc Alexa. 2021. The Diamond Laplacian for Polygonal and Polyhedral Meshes. In Computer Graphics Forum, Vol. 40. Wiley Online Library, 217–230.

Hugo Casquier, Yongjie Jessica Zhang, Carles Bona-Casas, Lisandro Dalcin, and Hector Gomez. 2018. Non-body-fitted fluid–structure interaction: Divergence-conforming B-splines, fully-implicit dynamics, and variational formulation. J. Comput. Phys. 374 (2018), 625–653.

Bernardo Cockburn, Fengyuan Li, and Chi-Wang Shu. 2004. Locally divergence-free quadratic discontinuous Galerkin methods for the Maxwell equations. J. Comput. Phys. 194, 2 (2004), 588–610.

Quaodong Cui, Timothy Langlois, Pradeep Sen, and Theodore Kim. 2021. Spiral-spectral fluid simulation. ACM Transactions on Graphics (TOG) 40, 6 (2021), 1–16.

Fang Da, David Hahn, Christopher Batty, Chris Wojtan, and Eitan Grinspun. 2016. Second-order-accurate schemes for magnetohydrodynamics with divergence-free reconstruction. ACM Transactions on Graphics (TOG) 35, 4 (2016), 1–11.

Tyler De Witt, Christian Lessig, and Eugene Fiume. 2012. Fluid simulation using Laplacian eigenfunctions. ACM Transactions on Graphics (TOG) 31, 2 (2012), 10.

Ivan DeWold. 2006. Divergence-free noise. Technical Report. Martian Labs., 2005.

Sharif Elcott, Yiying Tong, Eva Kanso, Peter Schröder, and Mathieu Desbrun. 2007. A hybridized divergence-free velocity field from grid-based data. In Proceedings of the ACM SIGGRAPH/Europographics symposium on Computer Animation. ACM, 237–246.

Alexis Angelidis and Fabrice Neyret. 2005. Simulation of smoke based on vortex filament primitives. In Proceedings of the 2005 ACM SIGGRAPH/Europographics symposium on Computer animation. ACM, 87–96.

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.

Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle. 2016. The material point method for simulating continuum materials. In ACM SIGGRAPH 2016 Courses 1–52.

Astrid Bunge, Mario Botsch, and Marc Alexa. 2021. The Diamond Laplacian for Polygonal and Polyhedral Meshes. In Computer Graphics Forum, Vol. 40. Wiley Online Library, 217–230.

Johnny Guzmán and Michael Neilan. 2014. Conforming and divergence-free Stokes elements in three dimensions. IMA J. Numer. Anal. 34, 4 (2014), 1489–1508.

Philip L. Fedder, Alexander Linke, Christian Merdon, and Joachim Schöberl. 2017. Divergence-free reconstruction operators for pressure-robust Stokes discretizations with continuous pressure finite elements. SIAM J. Numer. Anal. 55, 3 (2017), 1291–1314.

Christoph Lehrenfeld and Joachim Schöberl. 2016. High order exactly divergence-free hybrid discontinuous Galerkin methods for unsteady incompressible flows. Computer Methods in Applied Mechanics and Engineering 307 (2016), 339–361.

Johnny Guzmán and Michael Neilan. 2014. Conforming and divergence-free Stokes elements in three dimensions. IMA J. Numer. Anal. 34, 4 (2014), 1489–1508.

Colin P McNally. 2011. Divergence-free interpolation of vector fields from point values—exact div-B=0 in numerical simulations. Monthly Notices of the Royal Astronomical Society: Letters 413, 1 (2011), L76–L80.

TD Meyer and P. J. Froyen. 2004. Conservative velocity interpolation for PDF methods. In IASMM: Proceedings in Applied Mathematics and Mechanics, Vol. 4. Wiley Online Library, 466–467.

Yen Ting Ng, Chohong Min, and Frédéric Gibou. 2009. An Efficient Fluid-solid Coupling Algorithm for Single-phase Flows. J. Comput. Phys. 228, 23 (Dec. 2009), 8607–8629.

Colin P McNally. 2011. Divergence-free interpolation of vector fields from point values—exact div-B=0 in numerical simulations. Monthly Notices of the Royal Astronomical Society: Letters 413, 1 (2011), L76–L80.

DW Meyer and P. J. Froyen. 2004. Conservative velocity interpolation for PDF methods. In IASMM: Proceedings in Applied Mathematics and Mechanics, Vol. 4. Wiley Online Library, 466–467.

Tobias Pfaff, Nils Thueray, and Markus Gross. 2012. Lagrangian vortex sheets for animating fluids. ACM Transactions on Graphics (TOG) 31, 4 (2012), 112.

Charles S Peskin. 2002. The immersed boundary method. Acta numerica 11 (2002), 479–527.

Tobias Pfaff, Nils Thueray, and Markus Gross. 2012. Lagrangian vortex sheets for animating fluids. ACM Transactions on Graphics (TOG) 31, 4 (2012), 112.

Kostantin Poreikl and Konrad Pohlhir. 2016. Boundary-aware Hodge decompositions for piecewise constant vector fields. Computer-Aided Design 78 (2016), 126–136.

Amanda Pitsch, Boris P. Kaus, and Anton A. Popov. 2017. On the quality of velocity interpolation schemes for marker-in-cell method and staggered grids. Pure and Applied Geophysics 174, 3 (2017), 1071–1089.

Yannick Sauleau, Torsten Hardtch, Jorge Alejandro Amador Herrera, Daniel T. Banuti, Wojciech Palubicki, Soren Pirk, Klaus Hildebrandt, and Dominik L. Michels. 2022. DCGrid: An Adaptive Grid Structure for Memory-Constrained Fluid Simulation on the GPU. (2022).

Nick Rasmussen, Douglas Enright, Duc Nguyen, Sebastian Marino, Nigel Sumner, Willi Geiger, Samir Hoon, and Ronald Fedkiw. 2004. Detectable photorealistic liquids. In Proceedings of the 2004 ACM SIGGRAPH/Europographics symposium on Computer animation. 193–202.

Philip L. Fedder, Alexander Linke, Christian Merdon, and Joachim Schöberl. 2017. Divergence-free reconstruction operators for pressure-robust Stokes discretizations with continuous pressure finite elements. SIAM J. Numer. Anal. 55, 3 (2017), 1291–1314.

Syuhei Sato, Yoshinori Dobashi, and Theodore Kim. 2021. Stream-Guided Smoke Animation. ACM Transactions on Graphics (TOG) 40, 4, Article 161 (2021).
Syuhei Sato, Yoshinori Dobashi, and Tomoyuki Nishita. 2018a. Editing fluid animation using flow interpolation. *ACM Transactions on Graphics (TOG)* 37, 5 (2018), 1–12.

Syuhei Sato, Yoshinori Dobashi, Yonghao Yue, Kei Iwasaki, and Tomoyuki Nishita. 2015. Incompressibility-preserving deformation for fluid flows using vector potentials. *The Visual Computer* 31, 6 (2015), 959–965.

Takahiro Sato, Christopher Wojtan, Nils Thuerey, Takeo Igarashi, and Ryoichi Ando. 2018b. Extended narrow band FLIP for liquid simulations. In *Computer Graphics Forum*, Vol. 37. Wiley Online Library, 169–177.

Hagit Schechter and Robert Bridson. 2008. Evolving sub-grid turbulence for smoke animation. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association, 1–7.

Craig Schroeder, Ritoban Roy Chowdhury, and Tamar Shinar. 2022. Local divergence-free polynomial interpolation on MAC grids. *J. Comput. Phys.* 468 (2022), 115500.

Andrew Selle, Ronald Fedkiw, Byungmoon Kim, Yingjie Liu, and Jarek Rossignac. 2008. An unconditionally stable MacCormack method. *Journal of Scientific Computing* 35, 2-3 (2008), 350–371.

Han Shao, Libo Huang, and Dominik L. Michels. 2022. A Fast Unsmoothed Aggregation Algebraic Multigrid Framework for the Large-Scale Simulation of Incompressible Flow. *ACM Transaction on Graphics* 41, 4, Article 49 (07 2022).

Zachary J. Silberman, Thomas R. Adams, Joshua A. Faber, Zachariah B. Etienne, and Ian Ruchlin. 2019. Numerical generation of vector potentials from specified magnetic fields. *J. Comput. Phys.* 379 (2019), 421 – 437. https://doi.org/10.1016/j.jcp.2018.12.006

Jos Stam. 1999. Stable Fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH ’99)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 121–128. https://doi.org/10.1145/311535.311548

Jos Stam and Eugene Fiume. 1993. Turbulent wind fields for gaseous phenomena. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. 369–376.

Michael Steffen, Robert M Kirby, and Martin Berzins. 2008. Analysis and reduction of quadrature errors in the material point method (MPM). *International journal for numerical methods in engineering* 76, 6 (2008), 922–948.

Tetsuya Takahashi and Ming C Lin. 2019. A geometrically consistent viscous fluid solver with two-way fluid-solid coupling. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 49–58.

Yiying Tong, Santiago Lombeyda, Anil N Hirani, and Mathieu Desbrun. 2003. Discrete multiscale vector field decomposition. *ACM Transaction on Graphics (TOG)* 22, 3 (2003), 445–452.

Charles Van Loan. 1992. *Computational frameworks for the fast fourier transform*. SIAM.

Hongliang Wang, Roberto Agrusta, and Jeroen van Hunen. 2015. Advantages of a conservative velocity interpolation (CVI) scheme for particle-in-cell methods with application in geodynamic modeling. *Geochemistry, Geophysics, Geosystems* 16, 6 (2015), 2015–2023.

Mo Wang, Yiying Tong, Mathieu Desbrun, and Peter Schröder. 2006. Edge subdivision schemes and the construction of smooth vector fields. *ACM Transactions on Graphics (TOG)* 25, 3 (2006), 1041–1048.

Rundong Zhao, Mathieu Desbrun, Guo-Wei Wei, and Yiying Tong. 2019. 3D Hodge decompositions of edge-and face-based vector fields. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–13.

Yongning Zhu and Robert Bridson. 2005. Animating sand as a fluid. *ACM Transactions on Graphics (TOG)* 24, 3 (2005), 965–972.