

UC Riverside

UC Riverside Electronic Theses and Dissertations

Title

Divergence-Free Interpolation on Irregular Domains and Consistent Particle Initialization for Material Point Methods

Permalink

<https://escholarship.org/uc/item/0b93t24v>

ISBN

9798173719959

Author

Jin, Jing

Publication Date

2026-08-20

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA
RIVERSIDE

Divergence-Free Interpolation on Irregular Domains and Consistent Particle
Initialization for Material Point Methods

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Jing Jin

September 2026

Dissertation Committee:

Professor Craig Schroeder, Chairperson

Professor Evangelos Papalexakis

Professor Greg Ver Steeg

Professor Tamar Shinar

The Dissertation of Jing Jin is approved:

Committee Chairperson

University of California, Riverside

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor, Professor Craig Schroeder, for his invaluable guidance and support during my doctoral studies.

I would like to thank Craig Schroeder, my co-author on “Divergence-free interpolation of MAC grid data over irregular domains” [44] (Chapter 2), “Divergence-free interpolation of three-dimensional MAC grid data over irregular domains” (Chapter 3), and the particle initialization work for the Material Point Method (Chapter 4), for his contribution to this work.

I would also like to thank Professor Evangelos Papalexakis, Professor Greg Ver Steeg and Professor Tamar Shinar for serving on my dissertation committee and providing thoughtful insights and guidance.

I would like to thank my labmates Jason Goulding, Mehrnaz Ayazi, Song Bai, Chen Zhao, and Fox Liu for their helpful discussions and assistance throughout my time in the lab. I am especially grateful to Mehrnaz Ayazi for introducing me to the particle initialization project (Chapter 4) and for her help throughout that work.

The work in Chapter 2 was supported in part by National Science Foundation award NSF-2006570 and University of California award M23PL6076.

ABSTRACT OF THE DISSERTATION

Divergence-Free Interpolation on Irregular Domains and Consistent Particle Initialization
for Material Point Methods

by

Jing Jin

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, September 2026
Professor Craig Schroeder, Chairperson

Numerical simulations commonly store field data on a regular Eulerian grid, but the grid alone is rarely enough. For instance, particle tracing and semi-Lagrangian advection must evaluate the field at arbitrary locations. In those settings, accuracy depends on how faithfully discrete data and the continuum are connected, a connection most fragile where irregular boundaries intersect the grid data. This dissertation develops methods for two problems of this kind.

The first subject is the interpolation of divergence-free vector fields from MAC grid data over irregular domains. Preserving this property under interpolation ensures that the reconstructed field remains consistent with the incompressibility of the underlying flow. Spline-based divergence-free interpolation schemes are efficient on regular grids but break down near irregular boundaries, leaving a gap region in which the divergence-free property is lost. We construct interpolation schemes that fill this gap: over triangles and squares in two dimensions, and over tetrahedra and cubes in three dimensions, completing the incomplete divergence-free polynomial basis with rational functions. In three dimensions, cut

cells are divided by a specially constructed marching-cubes table producing non-inverted tetrahedra that match continuously across grid cells. The resulting interpolation is analytically divergence-free, C^0 continuous everywhere, local, and compatible with prescribed domain boundary velocities.

The second subject is the initialization of particle states for the Material Point Method, where inconsistent particle volumes and positions produce spurious startup motion which we refer to as the boiling effect. We express the inconsistency as a residual between particle-rasterized nodal volumes and target nodal volumes of the domain discretization and eliminate it before the first time step, either by repositioning the particles through a Gauss–Newton iteration or by adjusting their volumes through a regularized least-squares solve. The target volumes are computed in three ways: a radially symmetric closed-form approximation, exact tensor-product integration for box domains, and numerical quadrature. Experiments on square and cube domains show that their accuracy where walls meet is decisive: the exact and quadrature targets stop the boiling effect in two and three dimensions, whereas the radial approximation succeeds only in two.

TABLE OF CONTENTS

List of Figures	x
List of Tables	xviii
1 Introduction	1
1.1 Background	1
1.2 Divergence-Free Interpolation on Irregular Domains	3
1.3 Consistent Particle Initialization for the Material Point Method	5
1.4 Contributions	9
1.5 Dissertation Organization	11
2 Divergence-free Interpolation of MAC Grid Data Over Irregular Domains	12
2.1 Introduction	12
2.2 Discretization	18
2.2.1 Boundary representation	21
2.2.2 Canonical shapes	22
2.2.3 Basis selection	23
2.2.4 Triangles with cubic boundary polynomials	25
2.2.5 Squares	28
2.3 Cut-cell div-free interpolation	29
2.3.1 Cell classification	32
2.3.2 Interpolation in Interior cells	35
2.3.3 Node classification	35
2.3.4 Face classification	36
2.3.5 Face sample points	37
2.3.6 Summary of the Entire Interpolation Process	38
2.4 Analysis	40
2.4.1 Lipschitz continuity and bounds	40
2.4.2 Lipschitz continuity and bounds for triangles	42
2.5 Numerical tests	44
2.5.1 Random triangle and random square test	44
2.5.2 Visualization of div-free errors in canonical objects	46
2.5.3 Visualization of non-polynomial basis functions	47
2.5.4 Refinement Study	50
2.5.5 Streamline of the div-free velocity field	54

2.5.6	Full basis vs polynomial div-free basis	56
2.5.7	Path trace plotting	57
2.5.8	Random velocity field continuity test	59
2.5.9	Analytical velocity field projection test	62
2.6	Conclusion	64
3	Divergence-free Interpolation of Three-dimensional MAC Grid Data over Irregular Domains	65
3.1	Introduction	65
3.2	Discretization	72
3.2.1	Canonical shapes	73
3.2.2	Basis selection	77
3.3	Cut-cell div-free interpolation	77
3.3.1	Cell classification	79
3.3.2	Node classification	80
3.3.3	Edge classification	82
3.3.4	Face classification - cubes	82
3.3.5	Face classification - tetrahedra	87
3.3.6	Direct computation from transition faces	90
3.3.7	Interpolation procedure	92
3.4	Marching cubes with tetrahedra	93
3.4.1	Cube face triangulation	95
3.4.2	Tetrahedralization procedure - marching cubes	96
3.5	Numerical tests	100
3.5.1	Random tetrahedron and random cube test	101
3.5.2	Seeding MAC grid data	103
3.5.3	Refinement Study	104
3.5.4	Full basis vs polynomial div-free basis	108
3.5.5	Path trace plotting	110
3.5.6	Random velocity field continuity test	113
3.5.7	Analytical velocity field projection test	114
3.6	Conclusion	115
4	Reducing the Boiling Effect in the Material Point Method through Consistent Particle Initialization	125
4.1	Introduction	125
4.2	Discretization	131
4.2.1	Background	131
4.2.2	The Boiling Problem	133
4.2.3	Proposed Solution: Volume Matching	136
4.2.4	Solving the Matching Problem	137
4.3	Evaluating the Target Volume	143
4.4	Numerical tests	150
4.4.1	Setup	150
4.4.2	Parameter Settings	151

4.4.3	Comparison of Target Evaluation Methods	152
4.4.4	Volume Correction versus Position Correction	154
4.4.5	Node Volume Error Before and After Correction	156
4.4.6	Peak Velocity Evolution and Comparison to Lattice Seeding	160
4.4.7	Dam Break	163
4.4.8	Two-Density Stratification	165
4.5	Conclusion	168
4.5.1	Limitations and Future Work	169
5	Conclusion	171

LIST OF FIGURES

2.1	Canonical triangle (blue) with degree 2 and canonical triangle (red) with degree 3. Sample points occur at the vertices of the triangle and equally spaced along the edges. Along each edge this provides three or four sample points on each edge, which suffice to uniquely specify a quadratic or cubic polynomial along the edge. This polynomial specifies the value reconstructed field at the boundary.	20
2.2	Canonical square (blue) with degree 2 and canonical square (red) with degree 3. As with triangles, sample points for squares occur at the vertices of the square and equally spaced along the edges. Three points suffices to define a quadratic polynomial, and four points suffices for a cubic polynomial. . . .	20
2.3	Sample points for transition cells (●) are located at grid nodes (•) and grid faces (◐). Sample points for triangles within cut cells (○) are located at triangle vertices (•, ◐) and edges (◑).	32
2.4	(Left) Flux-averaged normal velocity samples are obtained along edges from MAC grid data (→), domain boundary conditions (→), and by requiring triangles to have zero net flux (→). (Right) Sample points are then computed by evaluating the spline-based interpolation scheme (•), evaluating the domain boundary conditions (•), using an integrated average of the domain boundary conditions (◐), interpolating from MAC grid data (•), and by fitting polynomials to the endpoints and flux-averaged normal velocity of an edge (•).	33

2.5	Cells in a domain that has been cut must be interpolated differently based on where they are with respect to the cut (—). (Left) Cells that are <i>outside</i> ($\square$) do not get interpolated. <i>Cut</i> cells ($\odot$), are cells that are cut by the boundary; these cells are triangulated and interpolated as triangles. Cells adjacent to cut cells are <i>transition</i> cells ($\bullet$); these cells have a neighboring cell that is cut. Because of the cut neighbor, the spline-based divergence-free interpolation scheme cannot be applied. These cells are interpolated as squares. Cells beyond the transition cells are classified as <i>interior</i> ($\blacksquare$). These are interpolated using the regular spline-based scheme. (Middle) Nodes are classified into four categories. <i>Interior</i> nodes ($\bullet$) are at corners at least one interior cell. <i>Outside</i> nodes ($\circ$) are outside the domain. <i>Cut</i> points ($\bullet$) are intersections of the domain boundary and faces. The rest are transition nodes ($\bullet$). (Right) Faces are be classified into <i>Interior</i> faces (—) that are shared by at least one <i>Interior</i> cell, <i>Boundary</i> faces (—) that connects two cut points, <i>Transition</i> faces which include other uncut faces (—) and <i>cut</i> faces(—), and <i>Auxiliary</i> faces (—) that are created to triangulate cut cells.	34
2.6	For each of the four combinations above, we constructed an interpolated velocity from random but flux-consistent sample values (in the sense that the total flux into the triangle is zero). Each of these velocity fields is analytically divergence-free everywhere. The plots show the central difference approximation to the divergence, colored using the color map shown. While the velocity field is analytically divergence-free everywhere, it is not C^1 at the corners. This causes the central difference approximation of the divergence errors to be much larger near the corners. (This figure is best viewed using the online version of this paper.)	46
2.7	The figure shows the two quadratic and three cubic non-polynomial basis functions for the canonical triangle. Here u is encoded as red and v is encoded as green, with the ranges mapped into the interval $[0, 1]$, as shown in the color legend. Observe that the colors are smoothly varying, a consequence of the Lipschitz continuity of the interpolation scheme. (This figure is best viewed using the online version of this paper.)	48
2.8	The figure shows the rational basis functions for the quadratic and cubic cases for the canonical square. Here u is encoded as red and v is encoded as green, with the ranges mapped into the interval $[0, 1]$. See Figure 2.7 for a color legend. Observe that the colors are smoothly varying, a consequence of the Lipschitz continuity of the interpolation scheme. (This figure is best viewed using the online version of this paper.)	49

2.9	This figure shows the value of $\ \nabla \mathbf{u}\ _F$ for the rational basis functions for the canonical triangle. For each rational basis, the values are mapped to $[0, 1]$. There are two such basis functions for the quadratic case and three for the cubic case. In the cubic case, each triangle has sharp velocity gradients at one vertex and is well-behaved elsewhere. The quadratic modes are each constructed from two of the cubic modes and thus show sharp gradients at two vertices.	50
2.10	This figure shows the value of $\ \nabla \mathbf{u}\ _F$ for the rational basis functions for the canonical square. For each rational basis, the values are remapped to $[0, 1]$. In each basis function, the velocity gradients are well-behaved except at one vertex.	51
2.11	This figure shows the ring-shaped domain used in the numerical tests. . . .	51
2.12	Divergence-free interpolation exhibits closed streamlines even in irregular flows through irregular domains at low resolution. Although the analytic velocity has zero normal component, the discrete boundary conditions are computed on a polygonal approximation of the circular boundaries and as such has nonzero normal components. This causes a few trajectories to exit the domain through the boundary.	54
2.13	The top illustration shows the different types of cells within a circular domain. The red diagonal line shows a path that crosses the interior region. This path visits all cell types as well as all types of transitions that can occur between types. The input velocity field of the domain is $\mathbf{u}_{2b}$, which is divergence-free but sufficiently high frequency that it cannot be resolved on this grid. The bottom-left plot shows the interpolated velocity along the red path when interpolating using the full bases. The red line indicates the u component, and the blue line indicates the v component. The interpolated velocity is continuous along the entire path. The bottom-right plot shows the interpolated velocity of the same path using only polynomial bases. This path contains discontinuities as the path transitions from one element to the next. The use of an unresolved data causes some of these discontinuities to be large enough to be visually observed. When interpolating smooth data, these discontinuities would be too small to visually observe.	58
3.1	The left figure shows the canonical tetrahedron with vertices at $(0, 0, 0)$, $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$. The right figure shows the canonical cube with vertices at $(0, 0, 0)$, $(1, 0, 0)$, $(0, 1, 0)$, $(1, 1, 0)$, $(0, 0, 1)$, $(1, 0, 1)$, $(0, 1, 1)$ and $(1, 1, 1)$. Vertex samples (●), edge samples (●), and face samples (●) are indicated.	74
3.2	Cell classification in a $3 \times 3 \times 1$ portion of the domain.	78

3.3	Velocities at transition nodes are reconstructed from neighboring face data. For each coordinate direction, the nodal normal velocity is obtained by a least-squares fit using four neighboring faces, whose MAC grid components are taken to live at the centroid of the interior portion of the grid face. For the emphasized direction, the interior portions of the face edges (—), the interior node (●), and exterior vertices (●) are shown. The four face centroids used in the fit are marked in green (●), and the node being computed is marked in black (●). The other two coordinate directions (.....) are omitted for clarity; in total, twelve faces are used for each node (four per velocity component).	80
3.4	Node classification within a $3 \times 3 \times 1$ region of the computational domain. (●) are interior nodes; these are nodes that touch interior cells. The nodes outside the domain are outside nodes (●). The cut points (●) are located on edges where they intersect the boundary. The rest are the transition nodes (●).	81
3.5	Classification of edges in a $3 \times 3 \times 1$ portion of the domain. transition cells. (—) indicates the interior edges, which are edges that touch an interior cell. (—) indicates the transition edges, which border a transition cell but not an interior cell. The diagonal edges (—) are diagonals of the square faces between the cut-cells and transition cells. Other edges inside the domain are cut edges (—).	83
3.6	(a) Ordering of samples on a transition face corresponding to (3.4). The black number indicates the index of corresponding matrix column for each input sample position; the green entry represents the integrated average across the square. The red number indicates the index of corresponding matrix row for each output face sample. (b) Ordering of samples on a triangle face corresponding to (3.5). The black number indicates the index of corresponding matrix column for each input sample position. The green entry is the integrated average across the triangle. The red number indicates the index of corresponding matrix row for each output face sample.	86
3.7	The square face between a transition cell and a cut cell is split along its diagonal into two triangles. This illustration shows the indexing convention used in Section 3.3.6 and shows the spatial location of sample points. The polynomial on the triangle is defined by interpolation from the neighboring transition cell that shares this face. Samples on the face and along the diagonal edge are computed from the velocities at the four square vertices and the average velocity integrated over the square face. Other edge samples are linearly interpolated.	91
3.8	For each marching cubes case, the marching cubes table encodes the boundary triangulation, surface triangulation, and interior tetrahedra.	94

3.9	This is an example of a forbidden tetrahedron under the non-negative volume requirement. The sign of the volume of this tetrahedron can change depending on where edges are cut. Such tetrahedra are not permitted in the marching cubes table.	95
3.10	Example of a polyhedron with its surface triangulation and boundary triangulation shown. Several boundary triangles are coplanar, which prevents the polyhedron from being tetrahedralized into valid nonnegative-volume tetrahedra using only the existing vertices. Additional vertices must be inserted to obtain a valid tetrahedralization. Polyhedra such as this one cannot be represented by a tetrahedralization in the marching-cube lookup table. The marching cubes table must be constructed to avoid cases such as this. . . .	95
3.11	Boundary triangulations for all 16 cases. For faces parallel to the xy plane or the xz plane, the x axis runs horizontally and the other axis runs vertically. For faces parallel to the yz plane, the y axis runs horizontally and the z axis runs vertically. The node with the lowest index is the bottom left vertex. The same case layout is used for all MAC grid faces, which causes the discretizations in neighboring cubes to be consistent.	99
3.12	Spherical shell domain used in the refinement study. The shell is centered at $(0.5, 0.5, 0.5)$ within the unit cube, with inner radius $r_{\text{in}} = 0.15$ and outer radius $r_{\text{out}} = 0.35$. The left panel shows a three-dimensional view of the inner and outer bounding spheres. The right panel shows a cross-section through the center, with the shell region shaded and the center and two radii marked.	106
3.13	Approximation error and divergence as functions of element conditioning. In both cases the maximum observed error correlates strongly with condition number κ of the tetrahedron. This shows that errors seen in L_∞ arise from poorly conditioned elements and are sensitive to element conditioning. These poorly conditioned elements typically have very small volumes.	108
3.14	Analytic divergence (the coefficient of the divergence basis mode) plotted against element volume (left) and element condition number (right). The magnitude is at most 10^{-10} , effectively zero. The larger values correlate strongly with tetrahedron volume rather than with condition number. . . .	109

3.15	This figure shows the interpolated velocity along a path through the domain. The path passes through cut cells and transition cells near the ends of the path and through the interior cells near the middle. The input velocity field of the domain is $\mathbf{u}_{3b}$, which is divergence-free but sufficiently high frequency that it cannot be resolved on this grid. The left plot shows the interpolated velocity along the path when interpolating using the full bases in u (top), v (middle), and w (bottom). The interpolated velocity is continuous along the entire path. The right plot shows the interpolated velocity of the same path using only polynomial bases. This path has discontinuities as the path transitions from one element to the next. This lack of continuity is caused by the lack of a full interpolation basis, which prevents the interpolated field from matching the prescribed boundary values and produces a discontinuity. The use of an unresolved data causes some of these discontinuities to be large enough to be visually observed. When interpolating smooth data, these discontinuities would be too small to visually observe. On full uncut MAC grid faces, the integrated average causes the high-frequency by very regular velocity field to be averaged out to near zero, resulting in a nearly constant interpolation in the interior.	112
4.1	The boiling effect. A body of fluid at rest under gravity, whose exact solution is $\mathbf{v} = \mathbf{0}$ for all time, is simulated with the natural particle initialization. (a) The initial state, at rest. (b) The boiling effect: the particles exhibit spurious velocities, colored by magnitude $ \mathbf{v}_p $. The agitation persists. . . .	134
4.2	Signed distance to a flat boundary. An interior node $\mathbf{x}_i$ ($d < 0$) retains most of its kernel support inside the domain, while an exterior node $\mathbf{x}_j$ ($d > 0$) contributes only a small spherical cap. In both cases the clipped integral is the same closed-form function $f(d)$, independent of the plane's orientation. . . .	144
4.3	Error of the radially symmetric closed form on a circle of radius $10h$ (2D) and a sphere of radius $4h$ (3D), against Gauss–Legendre quadrature of the tensor-product B-spline. Volumes are normalized by the unit cell volume, so an interior node has $V_{\text{quad}} = 1$. Left: volume difference versus signed distance. Right: relative error (lines, left axis) over nodes with reference volume above 10^{-3} , with the reference volume shown as bars (right axis). The largest relative errors occur where the reference volume is nearly zero, so their absolute contribution is small; the drop toward -100% past $d = \frac{3}{2}h$ reflects the spherical support assumed by the closed form.	147
4.4	Quadrature evaluation of the target volumes. A regular lattice places $q \times q$ points per grid cell (here $q = 2$; solid lines are grid cells of size h , dashed lines the quadrature subcells). Points outside the domain (open circles) are discarded by the level-set test; each remaining point (filled) carries an equal volume share $\Delta V_q = h^d/q^d$, and the targets are computed by the standard particle-to-grid gather over this set.	150

4.5	Hydrostatic test in 2D at water level 0.75 under the slip boundary condition, shown at frames 0, 30, 60, and 100 (left to right). (a) In the uncorrected baseline, visible flow patterns emerge and progressively disturb the particle distribution and the free surface. (b) With the position correction and quadrature-based target evaluation ($q = 2$), the particles stay at rest: the distribution remains uniform and the free surface stays flat.	155
4.6	Node volumes for the 2D hydrostatic test (unit box, water level 0.75, resolution 32), normalized by h^2 ; nodes with zero target are masked in gray. (a) Exact tensor-product targets, logarithmic scale; dashed lines mark the container walls and the free surface. (b) Signed error after Poisson-disk seeding, before correction. (c) Signed error after position correction; note the color scale differs from (b). (d) Absolute error after correction, logarithmic scale.	158
4.7	Residual energy $\frac{1}{2}\ \mathbf{r}(\mathbf{X}_k)\ ^2$ of the position correction versus Newton iteration on the hydrostatic test (resolution 32, exact targets), for three water levels in 2D (solid) and 3D (dashed). The 2D runs converge below the tolerance within nine iterations; the 3D runs stop at the iteration cap of ten with residuals near 10^{-5} , which is nevertheless sufficient to eliminate the artifact	159
4.8	Maximum velocity over time for the hydrostatic test on a 32^d grid, comparing target evaluation methods. The quadrature-based targets ($q = 2, 3$) are indistinguishable from the exact tensor-product targets, while the radial approximation and regular seeding retain residual motion.	161
4.9	Close-up of the initial particle distribution for the 2D hydrostatic test (unit box, water level 0.75, resolution 32). Each panel shows the same 8×8 -cell region of the domain, spanning the free surface: the dashed line marks the initial water level, with water below and empty cells above, and light gray lines indicate the grid cells. Left: regular lattice seeding with four particles per cell. Center: Poisson-disk sampling before correction. Right: the same Poisson-disk sample after position correction with quadrature-based targets ($q = 2$). The correction moves particles only slightly and preserves the irregular, evenly spaced character of the sample.	162
4.10	Dam break test: (a) initial configuration, (b) collapse, (c) impact against the right wall, (d) near rest. The water column of width 0.5 and height 1 is released in a box of height 2; frames from the uncorrected run.	163

4.11	Dam break test at resolution (32): (a) RMS and (b) maximum particle velocity over the first 1000 frames. The baseline uses the original particle initialization, while the corrected run uses the position fix with quadrature-based target evaluation ($q = 2$). The two runs show almost the same velocity behavior during collapse, impact, and later sloshing, indicating that the correction does not change the overall dynamics.	164
4.12	Sharp-interface two-density test: (a) root-mean-square and (b) maximum particle velocity over 300 frames, comparing baseline seeding with position correction using quadrature targets ($q = 2$). The exact solution is static. In the corrected run the initial settling oscillation decays steadily toward zero. In the baseline run the velocity remains large throughout: it does not decay monotonically but wanders, rising again after frame 250, as spurious motion continues to agitate the interface.	166
4.13	Particle distributions at frames 0, 30, 100, and 300 (left to right) for the sharp-interface two-density test; the light fluid is blue and the dense fluid is red. In the baseline run (a) the interface roughens and the particle arrangement in both layers is visibly disturbed. In the corrected run (b) the interface remains flat and the particle distribution is nearly unchanged over all 300 frames.	167

LIST OF TABLES

2.1	Stream functions and corresponding basis functions for the canonical triangle with quadratic polynomial boundary conditions. There are 9 divergence-free polynomial modes that are quadratic polynomials when evaluated at the boundary. We supplement these with two rational modes, which forms a full basis for the divergence-free subspace. Mode 12 is divergent and has no corresponding stream function; this mode completes our full basis. When the net flux across the boundary is zero, the coefficient for mode 12 will be zero, and the reconstructed result will be divergence free.	26
2.2	Stream functions and corresponding basis functions for the canonical triangle with cubic polynomial boundary conditions. The first 14 divergence-free modes are polynomials. We supplement these with 3 divergence-free rational modes to complete the divergence-free subset of the basis. Mode 18 is divergent and has no corresponding stream function.	26
2.3	Stream functions and corresponding basis functions for the canonical square with quadratic polynomial boundary conditions. There are 11 polynomial modes available, and we supplement them with four rational modes to complete the divergence-free subset of the basis. Mode 16 is divergent and has no corresponding stream function. The function $f(x, y)$ and its derivatives $f_x(x, y)$ and $f_y(x, y)$ are given in Table 2.4. The function $g(x, y)$ is constructed from $f(x, y)$ by adding polynomial terms that reduce the polynomials for the reconstructed fields at the boundary from cubics to quadratics. .	29
2.4	Stream functions and corresponding basis functions for the canonical square with cubic polynomial boundary conditions. In this case, we have 19 polynomial divergence-free modes available, which we supplement with four divergence-free rational modes. Mode 24 is divergent and has no corresponding stream function.	30
2.5	Summary of the Full Interpolation Scheme	39

2.6	The maximum value of $R(\mathbf{u}, \Omega)$ observed with different numbers of randomly sampled triangle-velocity points $(\Omega, \mathbf{u})$ varies wildly, takes on large values. The constant does not appear to be converging to any value, which suggests that the ratio is unbounded. The wild variations suggest that large values are possible but not especially common. In practice, these occur when a randomly generated triangle is especially poorly-conditioned.	42
2.7	This table shows the results of the numerical continuity tests. Here, “T” refers to triangles, “S” refers to squares; “Q” refers to quadratic polynomials and “C” refers to cubic polynomials; and “F” means the velocity field was divergence free and “D” means the velocity field was divergent. We observe continuity to roundoff error in all cases. Divergence is at the truncation error level for the central difference stencil used to evaluate the divergence error on the divergence-free cases.	45
2.8	This table shows the interpolation error in a ring shaped domain with velocity field $\mathbf{u}_{2a}$ and $\mathbf{u}_{2c}$ using the full interpolation basis. $\mathbf{u}_{2a}$ field is divergence-free while the $\mathbf{u}_{2c}$ field is divergent. “Ave” means the grid data was populated from the analytic velocity by averaging over the face. “Sample” means the velocity was obtained by sampling the analytic velocity at the face center. .	53
2.9	This table shows the interpolation error in a ring shaped domain with velocity field $\mathbf{u}_{2a}$ and $\mathbf{u}_{2c}$ using only polynomial basis functions. The $\mathbf{u}_{2a}$ field is divergence-free while the $\mathbf{u}_{2c}$ field is divergent. “Ave” means the grid data was populated from the analytic velocity by averaging over the face. “Sample” means the velocity was obtained by sampling the analytic velocity at the face center.	56
2.10	This table shows the results of projecting a divergence velocity field using the projection scheme from [63]. Interpolating this projected data yields a velocity field that is analytically divergence-free. The divergence errors are due to the truncation error in the central differencing divergence stencil, which is not changing under refinement. The errors show the difference between the interpolated projected velocity and the analytical projected result. The convergence order is approximately two.	63
3.1	Results of the numerical continuity tests. Here, “T” denotes tetrahedra and “C” denotes cubes. The labels “F” and “D” indicate whether the velocity field is divergence-free or divergent, respectively. In all cases, the continuity error on the boundary faces is at the level of roundoff. For the divergence-free cases, the measured divergence error is at the truncation-error level of the central-difference stencil used to compute it.	102

3.2	Interpolation error on a shell-shaped domain for velocity fields $\mathbf{u}_{3a}$, $\mathbf{u}_{3c}$, $\mathbf{u}_{3d}$ and $\mathbf{u}_{3e}$ using the full interpolation basis. Fields $\mathbf{u}_{3a}$ and $\mathbf{u}_{3d}$ are divergence-free, while $\mathbf{u}_{3c}$ and $\mathbf{u}_{3e}$ are divergent; $\mathbf{u}_{3d}$ and $\mathbf{u}_{3e}$ are affine. “Ave” means that each grid value stores the integrated average of the analytic velocity over the corresponding face. The results show that our interpolation scheme reproduces affine fields exactly and interpolates divergence-free fields to be point-wise divergence-free. The converge rate is about two in the ℓ_2 norm, while the ℓ_∞ norm exhibits noisier convergence around order two. The last column reports the maximum absolute analytic divergence (the coefficient of the divergence basis mode), which is near zero up to round-off error when the underlying velocity field is divergence-free.	105
3.3	Interpolation error on a shell-shaped domain for velocity field $\mathbf{u}_{3a}$, $\mathbf{u}_{3c}$, $\mathbf{u}_{3d}$ and $\mathbf{u}_{3e}$ using only the polynomial interpolation basis. $\mathbf{u}_{3a}$ and $\mathbf{u}_{3d}$ are divergence-free, while the $\mathbf{u}_{3c}$ and $\mathbf{u}_{3e}$ are divergent. $\mathbf{u}_{3d}$ and $\mathbf{u}_{3e}$ are affine. “Ave” means that each grid value stores the integrated average of the analytic velocity over the corresponding face. The results confirm that our interpolation scheme preserves divergence-free for both $\mathbf{u}_{3a}$ and $\mathbf{u}_{3d}$. Disregarding the discontinuity across elements, the polynomial basis attains a convergence rate comparable to the full basis as the grid is refined.	111
3.4	Results obtained by projecting a divergent velocity field using the projection scheme of [63]. Interpolating the projected data produces a velocity field that is analytically divergence-free. The reported divergence errors arise from the truncation error of the central-difference stencil used to evaluate the divergence, together with the effect of infrequent degenerate element cases; these errors do not decrease under refinement. The remaining errors measure the difference between the interpolated projected velocity field and the analytical projected solution. The observed convergence rate is approximately second order.	116
3.5	Polynomial basis modes for tetrahedron. The last mode (102) is the divergent mode. All of the other modes are divergence-free. Although there are a lot of polynomial basis functions, they are very efficient to compute.	117
3.6	Rational basis modes for tetrahedron.	118
3.7	Polynomial basis modes for cube.	120
3.8	Rational basis modes for cube.	122
4.1	v_{peak} and v_{rms} over 100 frames of the 2D hydrostatic test at resolution 32, for three water levels and two wall boundary conditions.	153

4.2	v_{peak} and v_{rms} over 100 frames of the 3D hydrostatic test at resolution 32, for three water levels and two wall boundary conditions.	154
4.3	Volume correction versus position correction on the hydrostatic test (resolution 32, slip boundary conditions, three water levels). All cases use quadrature-based target evaluation with $q = 2$	156

Chapter 1

Introduction

1.1 Background

Continuum physics describes materials ranging from fluids to solids through partial differential equations. Simulating these equations numerically has become an essential tool across science and engineering. In computer graphics, physics-based simulation produces visual effects such as snow [79], sand [47], and water that would be impractical to animate by hand. In computer science and engineering, the same machinery underlines the study of incompressible flow [22], magnetohydrodynamics [9], and geotechnical processes such as landslides and soil failure [73]. Across these applications, the central requirement remains the same: the discrete representation must faithfully preserve the underlying physics, while numerical artifacts introduced by the discretization must not be mistaken for genuine physical behavior.

Numerical simulation of continuum physics commonly stores field data on a regular Eulerian grid. Regular grids are efficient: the equations of motion can be discretized with

compact stencils, fast solvers apply directly, and global constraints such as incompressibility can be enforced through a projection step [22]. The grid alone, however, is rarely enough. semi-Lagrangian advection [75] and particle tracing [38] must evaluate the field at arbitrary locations within the domain, not merely at the grid points where data is stored. Hybrid grid-particle methods such as the Material Point Method (MPM) [80] go further. They represent the material itself as a set of Lagrangian particles carrying mass, volume, velocity, and other state information. And at every time step they transfer information between the particles and the grid. In both cases, the accuracy of the simulation depends not only on the grid solver but on how faithfully the discrete data and the continuum are connected.

This dissertation addresses two instances of this connection, each of which fails in a characteristic way when the boundary of the domain or of the material intersects the grid data. The first, and the main subject of this thesis, is the interpolation of divergence-free vector fields from MAC grid data over irregular domains: standard divergence-free interpolation schemes are exact on regular grids but break down near irregular boundaries, leaving a gap region in which the divergence-free property is lost (Section 1.2). The second is the initialization of particle states in MPM: common sampling strategies produce particle volumes and positions that are inconsistent with the underlying continuum representation, causing spurious motion at startup known as the boiling effect (Section 1.3). We introduce the necessary background and the specific challenges of each problem in turn.

1.2 Divergence-Free Interpolation on Irregular Domains

Many physical systems are governed by divergence-free vector fields (Equation (1.1)).

$$\nabla \cdot \mathbf{u} = 0. \tag{1.1}$$

In incompressible flow, the velocity field satisfies this constraint as an expression of volume conservation. The magnetic field in Maxwell’s equations and magnetohydrodynamics is subject to the same constraint, reflecting the absence of magnetic monopoles [9]. A common discretization for such fields is the marker-and-cell (MAC) grid [38], which stores the normal component of vector field at the center of each cell face. This staggered layout admits a natural discrete analogue of Eq. (1.1): a cell is discretely divergence-free when the fluxes across its faces sum to zero,

$$\sum_{f \in \text{faces}} u_f A_f = 0, \tag{1.2}$$

where u_f is the stored normal velocity on face f with respect to the outward normal, A_f is the face area (in a cut-cell discretization, the area of the portion of the face inside the domain [63]), and the product $u_f A_f$ is the flux of the vector field through that face. For an interpolated velocity field to be pointwise divergence-free, the interpolation must be built on data that satisfy this discrete condition. Grid data can be made discretely divergence-free by a projection step [22], which solves a Poisson problem for a pressure and subtracts its gradient from the velocity field.

Even when the grid data satisfy Eq. (1.2) exactly, however, the data live only at face centers, and querying the velocity at arbitrary points requires interpolation. There

the divergence-free structure is easily lost: interpolating each velocity component independently yields a field with nonzero divergence between grid points. The resulting errors are visible and accumulate over time. In flows with closed streamlines, the interpolated field spirals inward or outward instead of recirculating; traced particles drift across invariant flow structures; and volume is spuriously created or destroyed along trajectories [41, 20]. This has motivated a substantial body of work on interpolation schemes that are divergence-free by construction. On regular grids, local spline-based constructions achieve this efficiently: divergence-free splines date back to Handscomb [37], and recent constructions provide local, flux-consistent schemes of second and higher order [70, 67]. Alternatives based on radial basis functions [61, 57] or on reconstructing a vector potential [66, 72] generally require the solution of a global system, which makes the local grid-based constructions attractive in practice.

Most of these constructions, however, assume every cell is a full, uncut cell of a regular grid. Practical domains are rarely rectangles: fluid flows around immersed objects [64], and the domain boundary cuts through grid cells, producing cut cells with partial faces and a modified discrete divergence [63]. Near such boundaries the spline-based schemes break down for two reasons. First, they require a full neighborhood of uncut, discretely divergence-free cells, so they cannot be evaluated within a band of cells around the boundary. Second, the field near the boundary must be compatible with the velocity prescribed on the boundary itself, a constraint the regular-grid construction do not see. The result is a gap region between the domain boundary and the region where regular interpolation applies. Divergence-free interpolants for this regime do exist in the finite-element setting [35, 34], but they are built on simplicial meshes and their own degrees of freedom

rather than on MAC face data, and offer no direct remedy. In the gap region, then, no continuous divergence-free interpolant is available that is consistent with both the grid data and the boundary conditions. Filling this gap is harder than it may appear. Over individual triangles and squares (or tetrahedra or cubes in 3D), polynomials of a given degree that are divergence-free within each element cannot generally be matched continuously across element boundaries; achieving continuity requires enriching the space with non-polynomial functions [35, 34]. The difficulty is compounded in three dimensions, where cut cells take on a far greater variety of geometries than in the plane. Chapters 2 and 3 of this dissertation construct such interpolants in two and three dimensions, respectively. The resulting interpolants are local, C^0 continuous, pointwise divergence-free, and compatible with the prescribed boundary data.

1.3 Consistent Particle Initialization for the Material Point Method

The Material Point Method [80] simulates materials using particles and a background grid together. In MPM, the material is discretized into a set of particles, each representing a small portion of the material and moving with it. Each particle carries material state, including its mass m_p , volume V_p , velocity $\mathbf{v}_p$, and deformation gradient $\mathbf{F}_p$. The deformation gradient encodes local deformation history, from which the stress is evaluated through the constitutive model, the material-specific law relating deformation to stress. Meanwhile, the equations of motion are solved on the grid. Each time step transfers information from particles to grid nodes through the grid basis functions N_i , where $N_i(\mathbf{x}_p)$

weights the contribution of particle p to node i . Through this transfer, the grid accumulates mass, momentum, and other quantities from the particles; the equations of motion are then advanced on the grid, and the updated solution is transferred back to the particles. We sketch this cycle below in its simplest explicit form, following the formulation of Stomakhin et al. [79]; practical implementations refine each stage, but the structure remains the same.

The particle-to-grid transfer accumulates mass and momentum at the nodes, yielding the nodal mass m_i and momentum $m_i \mathbf{v}_i$, respectively:

$$m_i = \sum_p m_p N_i(\mathbf{x}_p), \quad m_i \mathbf{v}_i = \sum_p m_p \mathbf{v}_p N_i(\mathbf{x}_p), \quad (1.3)$$

The grid velocities ($\mathbf{v}_i$) are then advanced by the nodal forces ($\mathbf{f}_i$), which are evaluated by quadrature over the particles using the stress $\boldsymbol{\sigma}_p$ from the constitutive model.

$$\mathbf{v}_i^{\text{new}} = \mathbf{v}_i + \frac{\Delta t}{m_i} \mathbf{f}_i, \quad \mathbf{f}_i = - \sum_p V_p \boldsymbol{\sigma}_p \nabla N_i(\mathbf{x}_p), \quad (1.4)$$

where each particle contributes with weight V_p , its current volume. Finally, the updated velocities are interpolated back to the particles through the same basis functions, and the particles are advected,

$$\mathbf{v}_p^{\text{new}} = \sum_i \mathbf{v}_i^{\text{new}} N_i(\mathbf{x}_p), \quad \mathbf{x}_p^{\text{new}} = \mathbf{x}_p + \Delta t \mathbf{v}_p^{\text{new}}. \quad (1.5)$$

This sketch omits many components of a complete solver, but the underlying structure is what concerns us here: material state resides on the particles, forces and the momentum update are computed on the grid, and the two are coupled through the basis functions N_i .

The division of labor is what makes the hybrid scheme worthwhile. The particles provide a Lagrangian description of the material: mass and history-dependent state are carried with the motion, advection is handled trivially by moving the particles, and topological changes require no special treatment since the particles carry no connectivity. The grid, in turn, provides a regular Eulerian structure on which forces, collisions, and the momentum update are computed efficiently. Because the grid carries no persistent state, deformation never accumulates on it, allowing the grid to remain undistorted regardless of the material deformation. MPM therefore handles large deformation and complex material behavior robustly, and has been widely adopted in computer graphics [79] and computational mechanics [25].

Within this cycle, the particles also serve a second role: they are the quadrature points for the integrals in the governing equations. The internal force at a grid node,

$$\mathbf{f}_i = - \int_{\Omega} \boldsymbol{\sigma} \nabla N_i d\mathbf{x} \approx - \sum_p \boldsymbol{\sigma}_p \nabla N_i(\mathbf{x}_p) V_p, \quad (1.6)$$

is approximated by a weighted sum over the particles, with each particle's volume V_p serving as its quadrature weight [76]. This approximation is faithful only when the particle state is consistent with the grid and the domain: the volumes rasterized from particles onto each node must match the volume that node represents geometrically,

$$\sum_p V_p N_i(\mathbf{x}_p) \approx \int_{\Omega} N_i d\mathbf{x}, \quad (1.7)$$

where the right-hand side is the nodal volume of the domain discretization itself, and the quality of these targets limits how well any correction can perform. Equation (1.7) is a *local* condition, imposed at every node: the total particle volume may equal the domain volume exactly while individual nodes see too much or too little, and it is this local mismatch that corrupts the quadrature.

The consequence is visible at the very start of a simulation. A tank of fluid at rest under gravity should remain at rest, with internal forces balancing gravity exactly. When the initial particle state violates Eq. (1.7), however, the nodal forces evaluated from the particles no longer cancel gravity, and the material begins to move with no physical cause, resulting in a spurious, surface-concentrated agitation that we call the *boiling effect*. The violation has an interior component, where particle volumes must tile space correctly under the grid basis functions, and a boundary component, where the material region cuts through grid cells and particles claim too much or too little volume. Common sampling strategies such as jittered lattices and Poisson-disk sampling [19] produce distributions that look uniform but satisfy neither requirement.

Existing correcting techniques target different problems. Runtime corrections with MPM improve the particle–grid transfers [11, 42] or shift particles against density drift during the simulation [94, 13]; these operate at every time step and address errors as they arise, whereas the inconsistency considered here is present before the first step is taken. The closest precedent comes from smoothed particle hydrodynamics (SPH), a fully Lagrangian method in which field quantities are reconstructed directly from neighboring particles through smoothing kernels, with no background grid. Because the reconstruction quality depends on how the particles are arranged, SPH has developed particle packing

methods that adjust the initial configuration before the simulation begins [23, 62]. These methods, however, differ from the present setting in three respects: they target a kernel-consistency condition intrinsic to SPH rather than the grid condition of Eq. (1.7); they correct by iterating a pseudo-dynamic relaxation rather than by minimizing an explicit objective; and they adjust particle positions only, never treating particle volume as an unknown, even though in MPM the volume enters the quadrature directly as the weight. Chapter 4 of this dissertation formulates initialization for MPM itself: the violation of Eq. (1.7) becomes an explicit residual over the two unknowns of the quadrature, the particle volumes and the particle positions. Adjusting the volumes yields a regularized least-squares system; adjusting the positions, our preferred route, yields a nonlinear problem solved by Gauss–Newton. The correction runs once, before the first time step.

1.4 Contributions

This dissertation makes the following contributions.

- **Divergence-free interpolation of MAC grid data over irregular domains in two dimensions(Chapter 2).** We construct interpolation schemes on triangles and squares with arbitrary quadratic or cubic polynomial boundary data. Because divergence-free polynomials of these degrees do not form a complete basis on either shape, we augment them with rational functions. The resulting schemes are analytically divergence-free whenever the element boundary data have zero net flux and are continuous in all cases. Our construction fills the gap region between an irregular boundary and the interior, extending efficient spline-based divergence-free interpola-

tion [67] to cut-cell domains. The combined scheme is local, analytically divergence-free, and C^0 continuous throughout the domain, up to and including the domain boundary.

- **Extension to three dimensions (Chapter 3).** We present divergence-free interpolation schemes for cubes and tetrahedra: the cube scheme supports tensor-product cubic polynomial boundary conditions, and the tetrahedron scheme supports quartic polynomial boundary conditions. Cut cells are divided into tetrahedra by a specially constructed marching-cubes table that produces non-inverted tetrahedra consistent across neighboring grid cells. Applied on the cut cells, our schemes extend a spline-based divergence-free interpolation scheme to the boundary of an irregular domain; the resulting interpolation is C^0 continuous, analytically divergence-free, and remains purely local, built from polynomials and rational functions.
- **Consistent particle initialization for the Material Point Method (Chapter 4).** We cast the inconsistency behind the boiling effect as an explicit residual, Eq. (1.7), and drive it down by relocating particles with a Gauss–Newton solver, with a Krylov-based volume correction as the linear alternative. The procedure is a one-time preprocess requiring no change to the MPM time stepping. On square and cube domains, the accuracy of the target volumes at edges and corners proves decisive: exact targets stop the boiling in 2D and 3D, a radial approximation succeeds only in 2D, and a quadrature evaluation with a modest number of points performs identically to the exact computation while remaining applicable beyond the box. The method is developed and validated on box domains; irregular containers remain future work.

1.5 Dissertation Organization

The remainder of this dissertation is organized as follows. Chapter 2 presents the two-dimensional divergence-free interpolation scheme: C^0 continuous, analytical divergence-free interpolation over triangles and squares, used to extend spline-based interpolation over a MAC grid to irregular cut-cell domains. Chapter 3 extends the construction to three dimensions, with divergence-free bases for cubes and tetrahedra and a marching-cubes table for dividing cut cells. Chapter 4 addresses particle initialization for the Material Point Method, correcting particle volumes and positions to eliminate the boiling effect. This study is validated on box domains in two and three dimensions. Chapter 5 summarizes the contributions and discusses limitations and future work.

Chapter 2

Divergence-free Interpolation of MAC Grid Data Over Irregular Domains

2.1 Introduction

Divergence-free vector fields play an important role in the numerical simulation of many types of physical phenomena. They occur in Maxwell's equations and thus play a central role in simulations of magnetohydrodynamics [9, 29, 89, 4, 5, 6, 8]. They are also frequently used to represent velocity fields in simulations of the incompressible Navier Stokes [41, 10, 66, 55, 83] and Stokes equations [49, 45, 48]. One common approach to such simulations is to store physical quantities in a regular grid. The divergence-free property can then be enforced through a global projection step, which results in a field that is discretely

divergence-free. For many applications, this regular arrangement of data is sufficient. For other applications, however, these vector fields need to be interpolated to general locations within the simulation domain, such as for semi-Lagrangian advection or particle tracing. While there are many efficient ways to interpolate grid data, these do not generally preserve the divergence-free property. This has led to the development of a rich variety of schemes specifically designed to produce analytically divergence-free vector fields.

There are many approaches that have been proposed to address this problem. One popular approach is the use of radial basis functions (RBF). These methods generally require the solution of a global system [61, 54, 57]. Farrell et al. [30] uses a multi-level framework to improve the efficiency of the global method. Drake et al. [28] avoids the global system by constructing local approximants on separate subsets of samples, which are then combined to form the final result. Radial basis functions are commonly used for processing the unstructured data or scattered data [18, 3, 27, 15]. Yang et al. [89] proposes a similar strategy using a linear superposition of orthogonal functions and solving a global system. For multiresolution problems, wavelet-based approaches have also been explored [16, 82], including some with support for fluid domain boundary conditions [26, 77, 43]. Bittner and Urban [16] constructs divergence-free multi-wavelets based on cubic Hermite splines.

Another major class of methods constructs a vector potential and then utilizes the fact that the curl of a vector field is divergence-free [31, 66, 10, 39]. These methods usually require solving a large global system to compute the vector potential. Ravu et al. [66] uses spline functions for constructing the vector potential from grid data, and the coefficients of the spline are determined locally, achieving better efficiency than a global scheme. Silberman

et al. [72], Chang et al. [20] increase the efficiency by using a sweeping strategy, so that only the solution of a scalar Poisson problem for the vector potential is required. Rather than a global solve, Solin et al. [74] trains a network-based model incorporating a latent potential field variable.

Methods designed for regular grids typically produce an divergence-free interpolation over the space of polynomials. B-splines of appropriate orders naturally form a divergence-free polynomial reconstruction on regular grids. This was the basis for one of the earliest approaches [37], which used a global solve to construct natural cubic splines. Yassine and McGraw [90] instead constructs tensor-valued Bezier patches, where the control points for the patches are solved by a global system of equations in the least squares sense. Balsara [4] reconstructs the velocity field on adaptive mesh, resulting in a system of equations to solve for the coefficients of the quadratic polynomials. Balsara [5] then extended the method to unstructured data. Most of the grid-based polynomial divergence-free reconstructions, however, are local [58, 6]. Jenny et al. [41] developed a novel conservative interpolation scheme for collocated grids in two dimensions, which Wang et al. [85] extended to three dimensions and applied to steady state and time-dependent flow fields with strong velocity gradients. Tóth and Roe [81] propose an interpolant based on prolongation and restriction. Schroeder et al. [70] proposed a general construction for local divergence-free polynomial interpolation schemes with a focus on maintaining C^0 or C^1 continuity. Roy-Chowdhury et al. [67] extended this construction to fourth-order accurate interpolation and curl-free interpolation. They also proposed a flux-consistent interpolation scheme, which we use as part of our scheme.

Divergence-free interpolation on polygonal or polyhedral elements has received significant interest in the context of finite element or virtual element method discretizations [93, 12, 21]. One particularly popular polynomial construction is based on the use of Raviart-Thomas spaces [33, 86]. Frerichs and Merdon [33] introduces a local energy projector which commutes with the divergence operator. With the help of the operator onto a generalized Raviart-Thomas element space, they develop a piecewise divergence-free non-conforming virtual element method. In order to obtain the pressure-robustness for the div-free virtual element methods on polygonal meshes, Wei et al. [86] proposes a div-preserving reconstruction based on the Raviart-Thomas interpolation on local subtriangulations of the polygons. The finite element method with Raviart-Thomas spaces has been used for many applications [95]. Li and Rui [50] considers the Stokes problem on general simplicial meshes. The velocity space is the polynomial space enriched by the Raviart-Thomas-Nédélec space and achieves the exact div-free and pressure-independent velocity interpolation. Frachon et al. [32] studies the cut finite element discretizations of a Darcy interface problem based on the mixed finite element pairs on the space of discontinuous polynomial functions and the Raviart-Thomas space. Liu et al. [53] considers a quasi-regular background polygonal mesh and handles boundaries using cut cells. They produce a divergence-free velocity field everywhere except within an $O(h)$ neighborhood of the boundary. Balsara and Dumbser [7] develops a polynomial divergence-free reconstruction of the magnetic field up to fourth order accuracy for magnetohydrodynamics (MHD) applications. Their reconstruction is continuous in the normal component but not the tangential components. In general, these methods produce a divergence-free polynomial reconstruction of a vector field on polygo-

nal elements. Our reconstructions supplement polynomials with rational functions, which allows us to form a complete divergence-free basis.

Rather than using a mesh over the entire computation domain, as is normally done in finite element methods, one often prefers to use a regular grid in the interior for efficiency. In the presence of irregular boundaries, this leads to cut cells. Applying grid-based divergence-free interpolation over these cut cells does not generally lead to a divergence-free reconstruction within those cells. The problem of extending interpolation to irregular boundaries is thus of interest. The ability to handle irregular boundaries is of particular relevance to immersed boundary methods, where an immersed boundary (such as a rigid body [59], deformable body [64], or interfaces [71, 91]) is embedded in a grid. These methods are convenient since a regular grid may be used in most of the domain, with special treatment only required in or near cells which are cut by the body [84]; our proposed approach fits nicely within this framework. Prior work also has demonstrated the flexibility of such approaches. Chang et al. [20] supports irregular solid geometry by using marching squares to triangulate cut cells and then discretizes them using a novel curl-noise-inspired potential ramping procedure. Bao et al. [10] handles immersed boundaries and achieve divergence-free velocities using a vector potential-based formulation. They show that this leads to improved volume conservation. Kingora and Sadat-Hosseini [46] uses an interpolation-free immersed boundary method to simulate the incompressible flow problems with complex immersed boundaries; grid cells cut by the immersed boundary are reshaped to conform to the shape of the immersed boundary. While this method handles thin immersed objects, the discretization relies on an uncut grid that contains fluid everywhere. Albuquerque et al. [1] extends divergence-free interpolation for the immersed boundary to unstructured poly-

hedral grids. Like Chang et al. [20], we handle cut cells by triangulating them. However, we do so with a purely local method, without the need to construct a vector potential.

Most divergence-free methods are based on polynomial reconstructions. Polynomials are widely used because they are a convenient basis for reconstructions because they are easy to work with and have good numerical properties. However, the space of divergence-free polynomials over triangles or squares does not form a complete basis. This leads many divergence-free reconstructions over triangles to be discontinuous. Completing the basis requires including basis functions that are not polynomials. Notably, Guzmán and Neilan [36] constructed a continuous divergence-free interpolation over triangles in two dimensions by supplementing the usual polynomial basis functions with bubble functions, which are rational functions. Guzmán and Neilan [34] extended this basis to tetrahedra in three dimensions also using bubble functions. We propose a simpler set of rational functions to complete our basis for triangles. The two formulations are mathematically equivalent, however, as their basis elements can be expressed as a linear combination of ours (and visa versa). We also construct rational function basis functions to complete the divergence-free basis for squares.

In this paper we construct discretizations for triangles and squares that are C^0 continuous with quadratic or cubic polynomial data specified along their boundaries. The discretizations are analytically divergence free in the interior provided the net flux across the specified boundary is zero. This capability allows us to extend an existing divergence-free discretization intended for regular grids to handle irregular domains. We use our triangle and square discretizations to fill in the gap between the interior, which the regular scheme can efficiently handle, and the irregular boundary. The resulting discretization

will be C^0 continuous and analytically divergence-free everywhere. We demonstrate this end-to-end discretization on a cut MAC grid using method of Ng et al. [63] to project the velocity field to be discretely divergence-free, the flux-consistent spline-based divergence-free discretization of Roy-Chowdhury et al. [67] in the interior away from the boundary, and our new discretizations to fill in the boundary region. For the sake of terminology, we will assume throughout this paper that the quantity we are interpolating is a velocity field. We use the terms “face” and “edge” interchangeably.

2.2 Discretization

The intended application of our discretization is to extend spline-based divergence-free interpolation schemes, which can efficiently interpolate data from a regular grid, to handle irregular boundaries. We assume that the underlying grid data has been projected to be discretely divergence-free in the sense that the total flux into each cell is zero Ng et al. [63]. We also assume that the velocity is known analytically at the boundary and that this velocity field is a quadratic polynomial. In practice, this captures most common cases, such as stationary objects (zero), translating objects (constant) or rigid bodies (affine). In this paper, we refer to these boundary velocities as the *domain boundary conditions*. While a spline-based scheme could be used to interpolate velocities to the boundary, the velocity field would not be analytically divergence-free near the boundary. The interpolated velocity field would also not be compatible with the domain boundary conditions. This leaves us with a gap region between the domain boundary and the region over which the spline-based interpolation can be applied. This gap has prescribed boundary velocities, leading

to the need for our interpolation scheme to handle arbitrary polynomial element boundary conditions (up to some fixed polynomial degree). In practice, spline-based interpolation schemes require a neighborhood of uncut grid cells in order to produce an analytically divergence-free result, leading to a layer of uncut grid cells and a layer of cut grid cells, as shown in Figure 2.5. We thus construct a scheme for squares, which we apply to uncut grid cells, and triangles, which we used to discretize cut cells. We refer to the prescribed boundary velocities on these triangles and squares as *element boundary conditions*.

We begin by noting that, while it is always possible to be C^0 continuous with arbitrary polynomial boundary data, it will not always be possible to be analytically divergence-free, since

$$\int_{\partial\Omega} \mathbf{u} \cdot \mathbf{n} dA = \int_{\Omega} \nabla \cdot \mathbf{u} dV = 0 \quad (2.1)$$

is a constraint on the velocity at the boundary. If this constraint is satisfied, then we want to interpolate to a divergence-free velocity field. Otherwise, we are content to interpolate to a non-divergence-free velocity field, but in all cases we require continuity everywhere. Note that since the velocity field is specified along the entire boundary of each triangle or square, we can readily calculate the fluxes across each edge. The compatibility condition requires that these fluxes sum to zero.

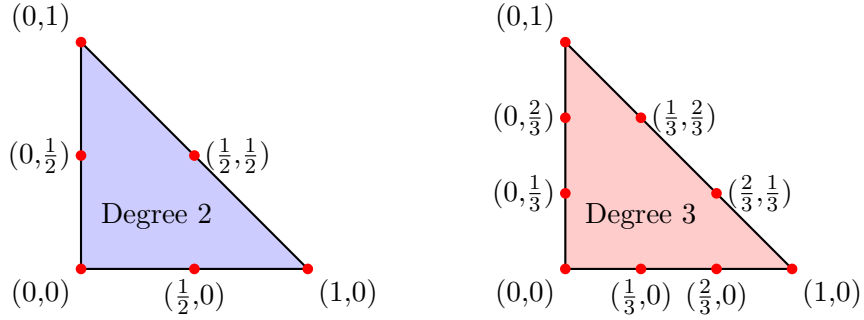


Figure 2.1: Canonical triangle (blue) with degree 2 and canonical triangle (red) with degree 3. Sample points occur at the vertices of the triangle and equally spaced along the edges. Along each edge this provides three or four sample points on each edge, which suffice to uniquely specify a quadratic or cubic polynomial along the edge. This polynomial specifies the value reconstructed field at the boundary.

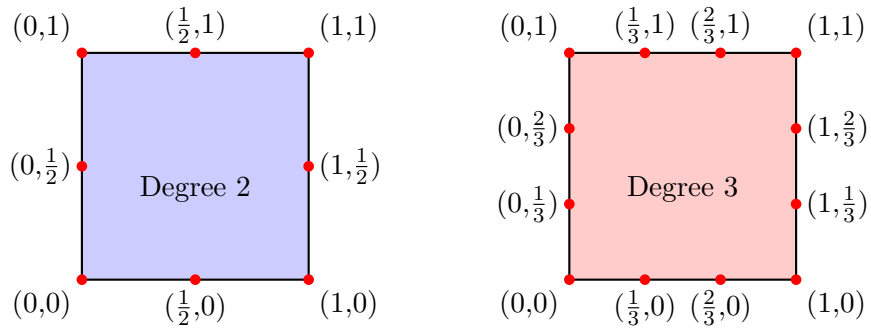


Figure 2.2: Canonical square (blue) with degree 2 and canonical square (red) with degree 3. As with triangles, sample points for squares occur at the vertices of the square and equally spaced along the edges. Three points suffices to define a quadratic polynomial, and four points suffices for a cubic polynomial.

2.2.1 Boundary representation

Although we assume that boundary data is a polynomial, representing the boundary data directly as polynomials is not very convenient since they are difficult to transform. Instead, we will represent our boundary velocity by sampling the boundary data at the vertices and at one (for quadratic polynomials) or two points (for cubic polynomials) equally spaced along each edge. This provides three or four *sample points* along each triangle or square edge (including the corners), which suffices to uniquely determine the original polynomial. The sample points are illustrated in Figures 2.1 and 2.2. We note that sample points are only used as a convenient representation for the polynomials that we specify on the boundary of triangles and squares; they do not in general coincide with MAC grid data. Sample points along an edge shared by adjacent shapes need not coincide, provided they are sampled from the same polynomial. We will use these sample points to construct the coefficients for our basis functions, which we use to perform interpolation within shapes.

Since there is a one-to-one correspondence between boundary velocity polynomials and sample points, the number of degrees of freedom N in the sample points is the same as the number of degrees of freedom available in the boundary polynomials themselves. The number of degrees of freedom is fixed ($N = 2sd$, where s is the number of sides (3 or 4) and d is the polynomial degree (2 or 3)). The problem of interpolation then reduces to the problem of selecting an independent set of $N - 1$ analytically divergence-free basis functions, which span the space of possible boundary polynomials that satisfy the flux condition. We can then supplement this basis with an extra divergent mode to ensure compatibility with the boundary when the boundary data has nonzero net flux.

2.2.2 Canonical shapes

Although the number of basis functions required is fixed, the basis functions themselves will depend on the shape and size of the triangles and squares being interpolated. Address this by constructing a suitable basis over a canonical triangle or square and then mapping this basis onto arbitrary triangles and squares. Our canonical triangle has vertices fixed at $(0, 0)$, $(1, 0)$, and $(0, 1)$, and our canonical square has vertices fixed at $(0, 0)$, $(1, 0)$, and $(1, 1)$, and $(0, 1)$. The canonical shapes are shown in Figures 2.1 and 2.2.

Let $\bar{\Omega}$ be our canonical shape, and let Ω be the original shape. Let $\phi : \bar{\Omega} \rightarrow \Omega$ be an *affine* mapping from the canonical shape to the original shape. That is, $\mathbf{x} = \phi(\bar{\mathbf{x}}) = \mathbf{A}\bar{\mathbf{x}} + \mathbf{b}$ for some matrix $\mathbf{A}$ and vector $\mathbf{b}$. (For triangles, $\mathbf{A}$ and $\mathbf{b}$ are easy to compute from the vertices of the original triangle. For squares, we only need translate and scale.) If $\bar{\mathbf{u}}(\bar{\mathbf{x}})$ is an interpolated velocity field in our canonical shape, then we can push this velocity forward to the original shape with

$$\mathbf{u}(\mathbf{x}) = \left(\frac{\partial \phi}{\partial \bar{\mathbf{x}}} \right) \bar{\mathbf{u}}(\bar{\mathbf{x}}) = \mathbf{A} \bar{\mathbf{u}}(\bar{\mathbf{x}}) \quad \bar{\mathbf{x}} = \phi^{-1}(\mathbf{x}). \quad (2.2)$$

Pushing forward a divergence-free vector field in this way preserves the divergence-free property, since

$$\frac{\partial \mathbf{u}(\mathbf{x})}{\partial \mathbf{x}} = \frac{\partial}{\partial \bar{\mathbf{x}}} (\mathbf{A} \bar{\mathbf{u}}(\bar{\mathbf{x}})) \frac{\partial \bar{\mathbf{x}}}{\partial \mathbf{x}} = \mathbf{A} \frac{\partial \bar{\mathbf{u}}}{\partial \bar{\mathbf{x}}} \mathbf{A}^{-1} \quad (2.3)$$

$$\nabla_{\mathbf{x}} \cdot \mathbf{u} = \text{tr} \left(\frac{\partial \mathbf{u}(\mathbf{x})}{\partial \mathbf{x}} \right) = \text{tr} \left(\mathbf{A} \frac{\partial \bar{\mathbf{u}}}{\partial \bar{\mathbf{x}}} \mathbf{A}^{-1} \right) = \text{tr} \left(\frac{\partial \bar{\mathbf{u}}}{\partial \bar{\mathbf{x}}} \right) = \nabla_{\bar{\mathbf{x}}} \cdot \bar{\mathbf{u}} = 0. \quad (2.4)$$

Here we used the fact that $\mathbf{A}$ is a constant, which was a consequence of ϕ being affine. This relationship is not true for more general mappings ϕ . We have now reduced our problem to the simpler task of finding a divergence-free basis over a canonical shape. Since these are fixed, once a basis is selected, it can simply be hardcoded.

One convenient advantage of using sample points rather than polynomials to represent the polynomial element boundary conditions is that it is straightforward to transform them into canonical space: $\bar{\mathbf{u}} = \mathbf{A}^{-1}\mathbf{u}$.

2.2.3 Basis selection

Next, we consider the basis selection process. Basis functions $\bar{\mathbf{u}}_k$ ($k = 1, 2, 3, \dots, N$), are velocity fields $\bar{\mathbf{u}}_k(x, y) = \langle u_k(x, y), v_k(x, y) \rangle$ that are polynomials of appropriate degree along the boundary of the canonical shape. Thus, for the triangle case $\bar{\mathbf{u}}_k(0, y)$, $\bar{\mathbf{u}}_k(x, 0)$, and $\bar{\mathbf{u}}_k(x, 1 - x)$ must be polynomials of appropriate degree (quadratic or cubic). For the square case, $\bar{\mathbf{u}}_k(0, y)$, $\bar{\mathbf{u}}_k(1, y)$, $\bar{\mathbf{u}}_k(x, 0)$, and $\bar{\mathbf{u}}_k(x, 1)$ must be polynomials of appropriate degree. Note that we do not require basis functions to be polynomials in the interior of the canonical shape.

Each function $\bar{\mathbf{u}}_k(x, y)$ can be associated with a *sample vector* $\hat{\mathbf{u}}_k$, which is the N -vector formed by evaluating the basis function at the locations of all of the $\frac{N}{2}$ sample points and stacking the results into one vector. We consider a set of basis functions $\{\bar{\mathbf{u}}_k\}$ to be linearly independent if their corresponding sample vectors $\{\hat{\mathbf{u}}_k\}$ are linearly independent. Note that if a set of basis functions is linearly dependent, then a linear combination of their corresponding sample vectors will be zero, which implies that the same linear combination of the basis functions will evaluate to the zero polynomial along the boundary of the canonical

shapes. While this linear combination of basis functions need not be zero in the interior, it has no effect on the boundary and is of no use to us.

A set of N linearly independent basis functions $\{\bar{\mathbf{u}}_1(x, y), \dots, \bar{\mathbf{u}}_N(x, y)\}$ forms a *full basis*; it spans the full space of possible element boundary conditions for the shape. We evaluate our interpolated velocity field as a linear combination

$$\bar{\mathbf{u}}(x, y) = \sum_{k=1}^N c_k \bar{\mathbf{u}}_k(x, y), \quad (2.5)$$

where the coefficients c_k must be chosen based on the specified boundary conditions for the canonical shape, which is specified by its sample vector $\hat{\mathbf{w}}$. This linear combination effectively defines interpolation within a single square or triangular canonical element. We can enforce continuity by requiring

$$\hat{\mathbf{w}} = \sum_{k=1}^N c_k \hat{\mathbf{u}}_k = \mathbf{K}\mathbf{c}, \quad (2.6)$$

where $\mathbf{K}$ is a matrix whose columns are $\hat{\mathbf{u}}_k$ and $\mathbf{c}$ is a vector of the coefficients c_k . We note that $\mathbf{K}$ is a fixed $N \times N$ matrix which will be known ahead of time and whose inverse can be precomputed and stored.

When constructing a basis, we will choose $\bar{\mathbf{u}}_1(x, y), \dots, \bar{\mathbf{u}}_{N-1}(x, y)$ to be divergence-free fields. The final basis function $\bar{\mathbf{u}}_N(x, y)$ must necessarily be divergent. For the divergent basis function, we select the affine mode $\hat{\mathbf{u}}_N = \langle x, y \rangle$, which allows the interpolation scheme to exactly reproduces affine functions, even if that affine function is not divergence free. This ensures that the interpolation will be second order accurate. If the element boundary

conditions are compatible with a divergence-free vector field, then its sample vector $\hat{\mathbf{w}}$ will be a linear combination of the divergence-free basis sample vectors $\bar{\mathbf{u}}_1(x, y), \dots, \bar{\mathbf{u}}_{N-1}(x, y)$. This will result in $c_N = 0$, ensuring that the divergent basis mode is not used to compute the interpolated vector field.

2.2.4 Triangles with cubic boundary polynomials

We are ready to begin constructing our interpolation basis functions. We begin by searching for a basis for the triangle case with cubic polynomials at the boundary. Since divergence-free vector fields are conveniently described by stream functions, we instead look for functions $s(x, y)$ and then let $u = -\frac{\partial s}{\partial y}$ and $v = \frac{\partial s}{\partial x}$. In this case, there are 9 sample points (three at the vertices and two per edge), which in 2D gives $N = 18$ degrees of freedom. Since one mode will be divergent, we need to find 17 stream functions. The obvious basis functions to use are polynomials. There are 14 independent polynomial stream functions whose corresponding u, v are cubic polynomials at the boundary: $x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3, x^4, x^3y, x^2y^2, xy^3, y^4$. There are in fact many more polynomials that are cubics at the boundary, such as $x^2y^2(1-x-y)^2$, which leads to basis functions that are identically zero at the boundary. The list of 14 is maximal, in the sense that any other polynomial would be linearly dependent with the ones above (that is, its sample vector is in the span of the 14 sample vectors for the polynomials listed). This leaves us 3 basis functions shy of a full basis.

One option would be to give up on completing the basis and instead solve (2.6) in a least squares sense. We evaluate the numerical properties of this strategy in Section 2.5.6.

Although the accuracy of this scheme is quite good, it cannot be continuous, since it cannot span the full space of cubic polynomial element boundary conditions.

basis	1	2	3	4	5	6	7	8	9
$s(x, y)$	x	y	x^2	xy	y^2	x^3	x^2y	xy^2	y^3
$u(x, y)$	0	-1	0	$-x$	$-2y$	0	$-x^2$	$-2xy$	$-3y^2$
$v(x, y)$	1	0	$2x$	y	0	$3x^2$	$2xy$	y^2	0

basis	10					11					12
$s(x, y)$	$\frac{x^3}{1-y} + \frac{x^3}{x+y} - x^3y - x^2y^2$					$\frac{x^3}{1-y} + \frac{y^3}{1-x} - x^3y - 2x^2y^2 - xy^3$					—
$u(x, y)$	$-\frac{x^3}{(1-y)^2} + \frac{x^3}{(x+y)^2} + x^3 + 2x^2y$					$-\frac{x^3}{(1-y)^2} - \frac{3y^2}{1-x} + x^3 + 4x^2y + 3xy^2$					x
$v(x, y)$	$\frac{3x^2}{1-y} - \frac{x^2(2x+3y)}{(x+y)^2} - 3x^2y - 2xy^2$					$\frac{3x^2}{1-y} + \frac{y^3}{(1-x)^2} - 3x^2y - 4xy^2 - y^3$					y

Table 2.1: Stream functions and corresponding basis functions for the canonical **triangle** with **quadratic** polynomial boundary conditions. There are 9 divergence-free polynomial modes that are quadratic polynomials when evaluated at the boundary. We supplement these with two rational modes, which forms a full basis for the divergence-free subspace. Mode 12 is divergent and has no corresponding stream function; this mode completes our full basis. When the net flux across the boundary is zero, the coefficient for mode 12 will be zero, and the reconstructed result will be divergence free.

basis	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$s(x, y)$	x	y	x^2	xy	y^2	x^3	x^2y	xy^2	y^3	x^4	x^3y	x^2y^2	xy^3	y^4
$u(x, y)$	0	-1	0	$-x$	$-2y$	0	$-x^2$	$-2xy$	$-3y^2$	0	$-x^3$	$-2x^2y$	$-3xy^2$	$-4y^3$
$v(x, y)$	1	0	$2x$	y	0	$3x^2$	$2xy$	y^2	0	$4x^3$	$3x^2y$	$2xy^2$	y^3	0

basis	15	16	17	18
$s(x, y)$	$\frac{y^3}{1-x}$	$\frac{x^3}{1-y}$	$\frac{x^3}{x+y}$	—
$u(x, y)$	$-\frac{3y^2}{1-x}$	$-\frac{x^3}{(1-y)^2}$	$\frac{x^3}{(x+y)^2}$	x
$v(x, y)$	$\frac{y^3}{(1-x)^2}$	$\frac{3x^2}{1-y}$	$-\frac{x^2(2x+3y)}{(x+y)^2}$	y

Table 2.2: Stream functions and corresponding basis functions for the canonical **triangle** with **cubic** polynomial boundary conditions. The first 14 divergence-free modes are polynomials. We supplement these with 3 divergence-free rational modes to complete the divergence-free subset of the basis. Mode 18 is divergent and has no corresponding stream function.

Instead, we must search for basis functions that are not polynomials. It turns out that there are suitable basis functions in the space of rational functions. One such stream

function is $s = \frac{x^3}{1-y}$, which leads to $u = -\frac{x^3}{(1-y)^2}$ and $v = \frac{3x^2}{1-y}$. These functions reduce to polynomials at the boundary since they vanish at $x = 0$, produce a constant denominator at $y = 0$, and $\frac{x}{1-y} = 1$ when $x + y = 1$. Additional basis functions can be found by exploiting symmetries of the triangle; the full basis is shown in Table 2.2.

While polynomials are trivially continuous, the same is not true for rational functions. In particular, the one above is potentially problematic when $1 - y = 0$, which occurs at one corner of the triangle. Let $r = \frac{x}{1-y}$. Since $0 \leq x \leq 1 - y$, we have $0 \leq r \leq 1$. Then $u = -(1 - y)r^3 \rightarrow 0$ and $v = 3(1 - y)r^2 \rightarrow 0$ as $y \rightarrow 1$. In this way, the basis function is continuous everywhere inside the triangle, though care should be taken when evaluating it near the corner. The other rational basis functions are symmetries of this one and have similar properties.

All that remains is to select the divergent mode. We select the affine mode $\langle x, y \rangle$, which ensures that the interpolation scheme will reproduce affine functions.

Quadratic boundary polynomials A basis for the quadratic case can be obtained from the cubic case since quadratic polynomials are also polynomials of degree at most 3. The general strategy is to select the polynomials that are of suitable degree and then to find linear combinations of the cubic basis functions using the rational elements that reduce to quadratics at the boundary. The corresponding basis is shown in Table 2.1. Note that only two rational basis functions are required, but they are both combinations of two of the cubic rational basis elements.

Note on differentiability Although the rational functions are continuous inside the triangle, their gradients are not. For example, for the mode $u = -\frac{x^3}{(1-y)^2}$ above, $\frac{\partial u}{\partial x} =$

$-\frac{3x^2}{(1-y)^2} = -3r^2$. Along the triangle edge $x = 0$, we have $\frac{\partial u}{\partial x} = 0$. Along the edge $y = 1 - x$, we have $\frac{\partial u}{\partial x} = -\frac{3x^2}{(1-y)^2} = -3$. Thus, the partial derivative is not continuous at the triangle corner. However, since $0 \leq r \leq 1$, the partial derivatives are all bounded. This implies that the basis functions are all Lipschitz continuous and thus well-behaved. Although the partial derivatives are not continuous, the divergence is identically zero everywhere and thus continuous.

2.2.5 Squares

The process of finding a suitable basis for the square is similar to the process for the triangle. In the cubic boundary polynomial case, we have $N = 24$ degrees of freedom at the boundary, of which 23 should be divergence-free. In this case the polynomials $s(x, y) = x^i y^k$ (with $i, k \leq 3$ and $i+k > 0$) as well as x^4 , y^4 , $\frac{1}{2}x^4 y^2(2y-3)$, and $\frac{1}{2}x^2 y^4(2x-3)$ are the admissible polynomials for the stream function, yielding 19 basis functions. We find that we are missing 4 basis functions. As with the triangle case, these functions can be chosen as rational functions. In this case, a suitable stream function is $s(x, y) = \frac{x^3(1-x)^2(1+x)^2}{x+y} - x^6 + x^5 y - x^4 y^2$. The rational piece has the property that its gradient manages to decay to a polynomial when $x = 0$, $x = 1$, $y = 0$, or $y = 1$. These polynomials may have degrees higher than cubic, so the extra polynomial terms are needed to reduce the boundary polynomials to cubics. The denominator vanishes at the corner of the square where $x = y = 0$. The rational function is a polynomial multiple of $\frac{x^3}{x+y}$, which has the property that it and its gradient both vanish in the limit $x, y \rightarrow 0$. As such, the stream function and velocity will be well-defined and continuous everywhere in the square. The qualitative behavior is similar to the triangle case.

As in the triangle case, the divergent mode is chosen to be $\langle x, y \rangle$ so that the interpolation scheme exactly reproduces affine functions, even if that affine function is not divergence free. The full bases for the quadratic and cubic polynomial cases are show in Tables 2.3 and 2.4. The continuity and differentiability properties of the rational modes for the square case are similar to those for the triangle case.

basis	1	2	3	4	5	6	7	8	9	10	11
$s(x, y)$	y	x	y^2	xy	x^2	xy^2	x^2y	x^2y^2	y^3	x^3	$2x^3y^3 - 3x^3y^2 - 3x^2y^3$
$u(x, y)$	-1	0	-2y	-x	0	-2xy	-x ²	-2x ² y	-3y ²	0	-6x ³ y ² + 6x ³ y + 9x ² y ²
$v(x, y)$	0	1	0	y	2x	y ²	2xy	2xy ²	0	3x ²	6x ² y ³ - 9x ² y ² - 6xy ³

basis	12	13	14	15	16
$s(x, y)$	$g(x, y)$	$g(x, 1 - y)$	$g(1 - x, 1 - y)$	$g(1 - x, y)$	-
$u(x, y)$	$-g_y(x, y)$	$g_y(x, 1 - y)$	$g_y(1 - x, 1 - y)$	$-g_y(1 - x, y)$	x
$v(x, y)$	$g_x(x, y)$	$g_x(x, 1 - y)$	$-g_x(1 - x, 1 - y)$	$-g_x(1 - x, y)$	y

$$g(x, y) = f(x, y) - 4x^3y + 3x^3y^2 + 4x^4 \quad g_y(x, y) = f_y(x, y) - 4x^3 + 6x^3y$$

$$g_x(x, y) = f_x(x, y) - 12x^2y + 9x^2y^2 + 16x^3$$

Table 2.3: Stream functions and corresponding basis functions for the canonical **square** with **quadratic** polynomial boundary conditions. There are 11 polynomial modes available, and we supplement them with four rational modes to complete the divergence-free subset of the basis. Mode 16 is divergent and has no corresponding stream function. The function $f(x, y)$ and its derivatives $f_x(x, y)$ and $f_y(x, y)$ are given in Table 2.4. The function $g(x, y)$ is constructed from $f(x, y)$ by adding polynomial terms that reduce the polynomials for the reconstructed fields at the boundary from cubics to quadratics.

2.3 Cut-cell div-free interpolation

In the section, we show how divergence-free interpolation for triangles and squares can be used to extend efficient spline-based interpolation schemes to handle irregular boundaries. For divergence-free interpolation to make sense, we must first start with grid data

basis	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$s(x, y)$	y	x	y^2	xy	x^2	xy^2	x^2y	x^2y^2	y^3	x^3	xy^3	x^3y	x^2y^3	x^3y^2
$u(x, y)$	-1	0	-2y	-x	0	-2xy	-x ²	-2x ² y	-3y ²	0	-3xy ²	-x ³	-3x ² y ²	-2x ³ y
$v(x, y)$	0	1	0	y	2x	y ²	2xy	2xy ²	0	3x ²	y ³	3x ² y	2xy ³	3x ² y ²

basis	15	16	17	18	19
$s(x, y)$	x^3y^3	x^4	y^4	$\frac{1}{2}x^4y^2(-3+2y)$	$\frac{1}{2}x^2y^4(-3+2x)$
$u(x, y)$	$-3x^3y^2$	0	$-4y^3$	$-x^4y(-3+2y) - x^4y^2$	$-2x^2y^3(-3+2x)$
$v(x, y)$	$3x^2y^3$	$4x^3$	0	$2x^3y^2(-3+2y)$	$xy^4(-3+2x) + x^2y^4$

basis	20	21	22	23	24
$s(x, y)$	$f(x, y)$	$f(x, 1-y)$	$f(1-x, 1-y)$	$f(1-x, y)$	-
$u(x, y)$	$-f_y(x, y)$	$f_y(x, 1-y)$	$f_y(1-x, 1-y)$	$-f_y(1-x, y)$	x
$v(x, y)$	$f_x(x, y)$	$f_x(x, 1-y)$	$-f_x(1-x, 1-y)$	$-f_x(1-x, y)$	y

$$\begin{aligned}
e(x, y) &= \frac{x(1-x)(1+x)}{x+y} & f(x, y) &= -2x^6 + 2x^5y - 2x^4y^2 + 2x(x+y)e(x, y)^2 \\
& & f_y(x, y) &= 2x^5 - 4x^4y - 2xe(x, y)^2 \\
& & f_x(x, y) &= -16x^5 + 10x^4y - 8x^3y^2 + 2x(3-7x^2)e(x, y) - 2xe(x, y)^2
\end{aligned}$$

Table 2.4: Stream functions and corresponding basis functions for the canonical **square** with **cubic** polynomial boundary conditions. In this case, we have 19 polynomial divergence-free modes available, which we supplement with four divergence-free rational modes. Mode 24 is divergent and has no corresponding stream function.

that has been projected to be discretely divergence-free. We use a MAC grid discretization for our regular grid, where the x and y components of velocity are stored at faces of cells.

For our notion of discretely divergence-free we follow the discretization of Ng et al. [63]. Here, cell faces through which the boundary passes are said to be cut, and the fraction of that face that is inside is called the *face fraction*. The face fraction s for uncut faces that are inside are 1, and the face fraction s for uncut faces that are outside are 0. For cut cells, it will be between 0 and 1. The discretization of Ng et al. [63] stores at MAC grid locations the normal component of the *integrated average* velocity over the interior portion of that face. The flux across a face is then the scalar velocity component stored at the face times the grid face size Δx times the face fraction. We consider a cell to be discretely divergence-free if the sum of fluxes into the cell is zero. Note that if nonzero velocities are specified at the boundary, the flux across the boundary in each cell also contributes to its divergence. This discretization of cut cell projection is convenient because of the simple relationship between the grid velocities and the definition of flux. See Figure 2.5.

A MAC grid provides only one scalar velocity value per grid face, but our divergence-free shapes require 2D velocities at sample points at their vertices and along their edges. It is also undesirable to use divergence-free shapes throughout the domain due to the computational cost. Instead, we want to use an efficient spline-based interpolation in the interior and then transition to shapes near the boundary. This requires us to classify grid cells depending on whether they will be interpolated with spline-based interpolation, a div-free square, or div-free triangles. We must then compute polynomial velocity fields of appropriate degree along the boundaries between shapes to ensure flux consistency and continuity. These boundary polynomials define the sample points used to compute basis coefficients for

the shapes. In this section, we construct a scheme that accomplishes all of these tasks. The procedure for computing the sample points is illustrated in Figure 2.4.

2.3.1 Cell classification

We begin by classifying cells based on the way they will be interpolated. Cells that are *outside* of the domain are not interpolated. These are uncut cells that are entirely outside of the boundary. Cells that have been cut by the boundary are called *cut* cells; cut cells are further divided into triangles using marching squares. These triangles will be interpolated using the quadratic version of the triangle scheme. Cells adjacent to cut cells we refer to as *transition* cells. These cells are not cut, but they are too close to the interface to apply the spline-based interpolation scheme. We will interpolate these cells using the cubic version of the square scheme. Beyond that we have *interior* cells, which we interpolate using the spline-based scheme. The classification is illustrated in Figure 2.5.

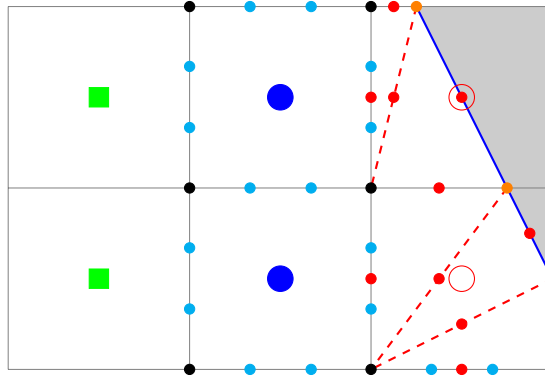


Figure 2.3: Sample points for transition cells (●) are located at grid nodes (•) and grid faces (•). Sample points for triangles within cut cells (○) are located at triangle vertices (•, •) and edges (•).

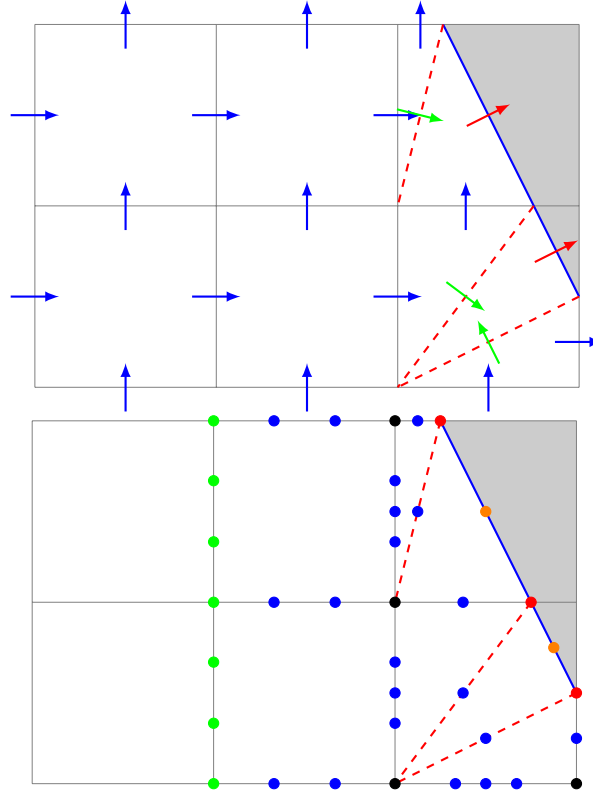


Figure 2.4: (Left) Flux-averaged normal velocity samples are obtained along edges from MAC grid data ($\rightarrow$), domain boundary conditions ($\rightarrow$), and by requiring triangles to have zero net flux ($\rightarrow$). (Right) Sample points are then computed by evaluating the spline-based interpolation scheme ($\bullet$), evaluating the domain boundary conditions ($\bullet$), using an integrated average of the domain boundary conditions ($\bullet$), interpolating from MAC grid data ($\bullet$), and by fitting polynomials to the endpoints and flux-averaged normal velocity of an edge ($\bullet$).

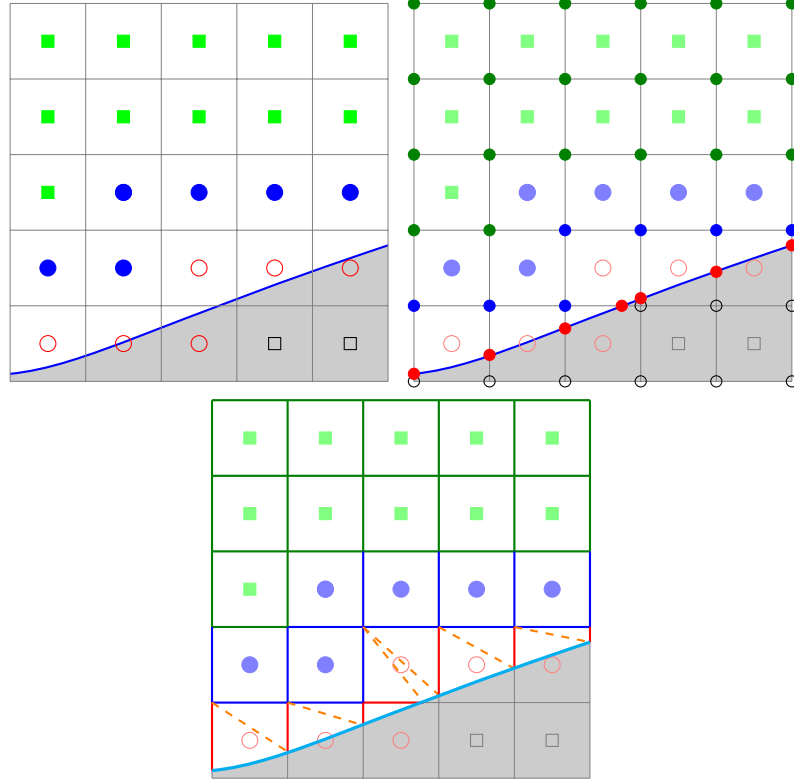


Figure 2.5: Cells in a domain that has been cut must be interpolated differently based on where they are with respect to the cut (—). (Left) Cells that are *outside* ($\square$) do not get interpolated. *Cut* cells ($\circ$), are cells that are cut by the boundary; these cells are triangulated and interpolated as triangles. Cells adjacent to cut cells are *transition* cells ($\bullet$); these cells have a neighboring cell that is cut. Because of the cut neighbor, the spline-based divergence-free interpolation scheme cannot be applied. These cells are interpolated as squares. Cells beyond the transition cells are classified as *interior* ($\blacksquare$). These are interpolated using the regular spline-based scheme. (Middle) Nodes are classified into four categories. *Interior* nodes ($\bullet$) are at corners at least one interior cell. *Outside* nodes ($\circ$) are outside the domain. *Cut* points ($\bullet$) are intersections of the domain boundary and faces. The rest are transition nodes ($\bullet$). (Right) Faces are be classified into *Interior* faces (—) that are shared by at least one *Interior* cell, *Boundary* faces (—) that connects two cut points, *Transition* faces which include other uncut faces (—) and *cut* faces (—), and *Auxiliary* faces (---) that are created to triangulate cut cells.

2.3.2 Interpolation in Interior cells

Efficient spline-based divergence-free interpolation schemes exist for regular MAC grid layouts [37, 70, 67]. These schemes are only usable when the MAC cells they collect data from have been discretely projected in a way that is consistent with a uncut MAC grid. Since cut cells have a different discrete notion of divergence, these interpolation schemes will produce divergence velocity fields if evaluated too close to the interface. We must therefore take care not to use them too close to the interface.

Another consideration when choosing an interpolation scheme for the interior is flux consistency. After projection using Ng et al. [63], MAC grid locations store the normal component of the integrated average velocity across the interior portion of the grid face. The flux across the face can be computed by multiplying the velocity stored on the MAC grid by the grid size and face fraction. The output of an interpolation scheme is a velocity field defined at all points in the domain; integrating this velocity field over a grid face produces another value for the flux across the face. If these two fluxes agree, we call the interpolation scheme *flux consistent*. In this paper, we use the flux consistent scheme from [67] to interpolate velocity in interior cells.

2.3.3 Node classification

The discretization of transition and cut cells must be done carefully so that we maintain continuity between neighbors. The easiest way to ensure this consistency is to compute velocities at nodes and faces first. To do this, we must classify nodes. *Interior nodes* are nodes that are the corner of at least one interior cell. The velocity for these nodes can be computed by evaluating the spline-based interpolation scheme at the location of the

node. *Outside nodes* are nodes that lie outside the interface. These nodes play no role since they are never used. All other grid nodes are classified as *transition nodes*; in almost all cases, these nodes are the corner of at least one transition cell but not the corner of an interior cell, though in degenerate cases it is possible for a transition node to have only cut cells as neighbors. We compute the velocity at transition nodes as

$$\mathbf{u}_{i,j} = \left\langle \frac{f_{i+\frac{1}{2},j}u_{i-\frac{1}{2},j} + f_{i-\frac{1}{2},j}u_{i+\frac{1}{2},j}}{f_{i-\frac{1}{2},j} + f_{i+\frac{1}{2},j}}, \frac{f_{i,j+\frac{1}{2}}v_{i,j-\frac{1}{2}} + f_{i,j-\frac{1}{2}}v_{i,j+\frac{1}{2}}}{f_{i,j-\frac{1}{2}} + f_{i,j+\frac{1}{2}}} \right\rangle, \quad (2.7)$$

where $f_{i+\frac{1}{2},j}$ denotes the face fraction along a face. In addition to grid nodes, we introduce *cut points* along cut faces at the point where they are cut. The velocities of cut points are dictated by boundary conditions. The node classification is shown in Figure 2.5.

2.3.4 Face classification

With nodal velocities chosen, we are ready to choose face velocities. Since our interpolation schemes use sample points, we must compute cut points for each face that will participate in triangle or square interpolation. As with nodes, we do this by first classifying faces. Because [67] produces cubic polynomials at faces, we must use the cubic square interpolation discretization. We assume that our cut boundary is defined by quadratic polynomials, which allows us to use the simpler quadratic discretization for triangles. Faces adjacent to cut cells require one sample point in the middle of the inside portion of each face. Faces adjacent to transition cells require two equally spaced cut cells at $\frac{1}{3}$ and $\frac{2}{3}$ along the face. Note that many faces shared by a cut cell and a transition cell will end up with

three sample points. The face classification is shown in Figure 2.5. The layout of sample points within the grid is shown in Figure 2.3.

Interior faces are faces of interior cells; their velocities are dictated by the spline-based interpolation scheme; sample points for these faces are computed by evaluating the interpolation scheme at the locations of the sample points. We connect adjacent cut points along the boundary with segments that we call *Boundary faces*. These are not faces of the MAC grid, but they participate in triangle interpolation within cut cells and require a sample point at their midpoint, which can be evaluated from domain boundary conditions. We refer to the inside portions of grid faces that have been cut as cut faces. The remaining faces adjacent to transition cells or cut cells we refer to as *transition faces*. When we divided cut cells into triangles using marching squares, we created some faces between the triangles. We call these *auxiliary faces*.

2.3.5 Face sample points

We compute sample points for cut faces, transition faces, and auxiliary faces using the velocity samples at the endpoints along with the integrated average normal velocity component. In the case of cut faces and transition faces, the integrated average component is just the scalar MAC velocity sample stored on that face. The integrated average normal velocity component for auxiliary faces is computed by requiring the total flux into each triangle to be zero. The endpoints lie at grid nodes or cut points; the velocities at those locations were computed in Section 2.3.3.

We define the velocity field along the face separately for the normal and tangential components. In each case, we let u_a and u_b be the component of velocity at the endpoints. In the tangential direction, we define the velocity profile along the face to be

$$u(t) = u_a + t(u_b - u_a), \quad (2.8)$$

where $t = 0, 1$ are the endpoints of the face. The normal direction must be treated more carefully, since it must be consistent with the integrated average normal velocity u_m . We define the normal velocity along a face to be the quadratic polynomial

$$u(t) = 3(u_a + u_b - 2u_m)t^2 - 2(2u_a + u_b - 3u_m)t + u_a. \quad (2.9)$$

Finally, sample points are computed by evaluating the polynomials for the normal and tangential components at the locations of the sample points.

Non-divergence-free interpolation Although the intended application of this interpolation scheme is for constructing an analytically divergence-free interpolated velocity field, we still would like the method to be well-behaved even if this is not possible. When a cut cell has nonzero divergence, we compute fluxes across auxiliary faces such that this divergence is distributed among the triangles inside the cut cell proportional to their areas.

2.3.6 Summary of the Entire Interpolation Process

In short, our objective is to construct the interpolant that handles the irregular boundary on a MAC grid. The input MAC data are the velocities stored at the face centers

along with the function for evaluating the velocity of the domain boundary. Based on those given data, we achieve our goal by discretizing the whole domain as triangles and squares. And to do interpolation inside each element we collect sample points along the element boundaries. This whole procedure is shown in Table 2.5.

Table 2.5: Summary of the Full Interpolation Scheme

1:	procedure PRECOMPUTE
2:	Classify cells
3:	Classify nodes and compute node velocities
4:	Classify faces and compute face velocities
5:	Precompute transition cells
6:	Collect sample points
7:	Compute coefficients of the basis
8:	Precompute cut cells:
9:	Make triangles
10:	Compute flux on shared edges of the triangles
11:	Collect sample points
12:	Compute coefficients of the basis
13:	end procedure
14:	procedure INTERPOLATE
15:	Find the cell containing the query point
16:	if interior cell then
17:	Apply flux-consistent spline-based interpolation
18:	else if transition cell then
19:	Compute square basis values
20:	Scale with precomputed coefficients
21:	else if cut cell then
22:	Find the triangle containing the query point
23:	Compute triangle basis values
24:	Scale with precomputed coefficients
25:	end if
26:	end procedure

2.4 Analysis

2.4.1 Lipschitz continuity and bounds

Let Ω be a region of space and $\mathbf{u}(\mathbf{x})$ be a vector field. Then, we can define the Lipschitz constant $L(\mathbf{u}, \Omega)$ as

$$L(\mathbf{u}, \Omega) = \sup_{\mathbf{x}, \mathbf{y} \in \Omega} \frac{\|\mathbf{u}(\mathbf{x}) - \mathbf{u}(\mathbf{y})\|}{\|\mathbf{x} - \mathbf{y}\|}. \quad (2.10)$$

The constant $L(\mathbf{u}, \Omega)$ gives us a measure of how wildly varying the vector field $\mathbf{u}$ is. In the interior, our basis functions are differentiable, and we can instead use the more computationally efficient form

$$L(\mathbf{u}, \Omega) = \sup_{\mathbf{x} \in \Omega} \|\nabla \mathbf{u}(\mathbf{x})\|_2. \quad (2.11)$$

If the specified boundary velocity values are wildly varying, then the interpolated velocity values necessarily will be as well. However, if the boundary velocity values are well-behaved, then the interpolated velocity values ought to be as well. Let Ω be a triangle or a square, and let $\partial\Omega$ be its boundary. Then the ratio

$$R(\mathbf{u}, \Omega) = \frac{L(\mathbf{u}, \Omega)}{L(\mathbf{u}, \partial\Omega)} \quad R(\Omega) = \sup_{\mathbf{u}} R(\mathbf{u}, \Omega) \quad (2.12)$$

captures how well-behaved the interpolation scheme is for a fixed geometry. Note that $L(\mathbf{u}, \Omega)$ can be computed from (2.11) but $L(\mathbf{u}, \partial\Omega)$ must be computed from (2.10). Since $\partial\Omega \subset \Omega$, we necessarily have $R(\Omega) \geq 1$. Note that the denominator is computed by

considering any two points on the boundary of the square or triangle, including points that are on different edges. A natural question to ask is whether $R(\Omega)$ is bounded over all possible choices of $\mathbf{u}$ and Ω . If Ω is the canonical triangle or square, then this is clearly the case. The basis functions are Lipschitz continuous, and the coefficients for these basis functions are obtained by solving (2.6), which amounts to multiplying the sample points by the fixed matrix $\mathbf{K}^{-1}$. Lipschitz continuity is thus guaranteed. For the general case, we explore the Lipschitz bounds numerically.

Numerical estimation of $L(\mathbf{u}, \Omega)$ Although direct estimation from (2.11) is straightforward, it converges very slowly. Instead, we estimate the using coordinate descent for the optimization. We try all locations on a regular 10×10 grid in the space of the canonical triangle or square as initial points for the optimization so that the global maximum is very likely to be found.

Numerical estimation of $L(\mathbf{u}, \partial\Omega)$ The boundary case in general requires us to use (2.10), which is less efficient because we must now sample at two locations. In practice, we observe that the maximum is achieved at one of two simpler cases, so we compute them separately. The case of both sample points being on one edge amounts to computing the maximum magnitude of the derivative of a cubic polynomial, which we solve analytically. The second special case is when the two sample points are arbitrarily close to a vertex but on different edges. This reduces to singular value problem, which we solve analytically. What is left is the general case when both points are on different edges. Since each point's location is along an edge, it is given by a single scalar. Thus, we must optimize (2.10) over

two scalars. We perform this optimization using coordinate descent and initialization over a 10×10 as in the interior case.

In both the boundary and interior cases, we must still sample over different choices of Ω and $\mathbf{u}$. Once these have been chosen, we are able to use the procedures above to accurately and efficiently compute the corresponding Lipschitz bounds and then calculate $R(\mathbf{u}, \Omega)$.

Square In the case of the square, the only freedom in choosing Ω is location and scale; since $R(\Omega)$ is invariant on these transformations, $R(\Omega)$ is just a fixed number. For $n = 10^3, 10^4, 10^5$ we estimate $R(\Omega)$ by sampling n different $\mathbf{u}$ and Ω using cubic polynomials. The corresponding numerical estimates for $R(\Omega)$ are 3.863, 4.245, and 4.454. This suggests that $R(\Omega)$ is bounded.

2.4.2 Lipschitz continuity and bounds for triangles

m	Max $R(\mathbf{u}, \Omega)$	
	Quadratic	Cubic
100	5.41×10^2	4.77×10^1
200	9.52×10^1	9.86×10^1
400	6.76×10^3	9.74×10^1
800	1.16×10^3	1.15×10^3
1600	6.81×10^2	4.69×10^3
3200	9.46×10^3	9.35×10^2
6400	8.19×10^3	7.78×10^2
12800	6.13×10^4	1.20×10^5

Table 2.6: The maximum value of $R(\mathbf{u}, \Omega)$ observed with different numbers of randomly sampled triangle-velocity points ($\Omega, \mathbf{u}$) varies wildly, takes on large values. The constant does not appear to be converging to any value, which suggests that the ratio is unbounded. The wild variations suggest that large values are possible but not especially common. In practice, these occur when a randomly generated triangle is especially poorly-conditioned.

The case of triangles is much more complex than the case for squares. To analyze it numerically, we randomly sample $(\Omega, \mathbf{u})$ pairs and then estimate $R(\mathbf{u}, \Omega)$ numerically as described above. As in the square case, we can get a sense for whether $R(\mathbf{u}, \Omega)$ is bounded by adding additional samples. The results of this experiment are shown in Table 2.6. Unlike in the square case, the ratios observed may be quite large (over 10^3) and do not appear to be converging. This suggests that there might not be a finite bound. Doubling the number of samples often does not reveal a new maximum, suggesting that large values may be unlikely in practice. Investigating triangles that produced particularly large ratios leads to the observation that large ratios correspond to poorly conditioned triangles.

Indeed, consider a triangle with vertices $(0, 0)$, $(1, 0)$, and $(-1, \epsilon)$, where $\epsilon > 0$ is small. The velocity field

$$\mathbf{u} = \begin{pmatrix} 2(\epsilon x - \epsilon + 2y)(\epsilon x + y)y/\epsilon^3 \\ (-x^2\epsilon^3 - (2x - 1)y^2\epsilon - 2y^3)/\epsilon^2 \end{pmatrix} \quad (2.13)$$

is a cubic polynomial and divergence free, and so the interpolation scheme with cubic polynomials will exactly recover this vector field in the interior. Asymptotically, we find $L(\mathbf{u}, \Omega) \rightarrow 1/\epsilon$ and $L(\mathbf{u}, \partial\Omega) \rightarrow 2$, so that $R(\mathbf{u}, \Omega) \rightarrow 2/\epsilon$. That is, *even with just polynomial basis functions*, $R(\Omega)$ is unbounded. However, if the triangles are well-conditioned, then $R(\Omega)$ will be bounded. This suggests that large Lipschitz constants in poorly-conditioned elements are a consequence of the elements themselves, not a limitation in the way the divergence-free interpolation was constructed. Achieving a bounded Lipschitz ratio is thus a matter of avoiding poorly conditioned triangles.

2.5 Numerical tests

We evaluate the performance of the proposed interpolation scheme using a variety of numerical tests. In these tests, we use the vector fields from [67]:

$$\begin{aligned}\mathbf{u}_{2a} &= \begin{pmatrix} \sin(x+2)\sin(y+4) \\ \cos(x+2)\cos(y+4) \end{pmatrix} & \mathbf{u}_{2b} &= \begin{pmatrix} \sin(370x+2)\sin(370y+4) \\ \cos(370x+2)\cos(370y+4) \end{pmatrix} \\ \mathbf{u}_{2c} &= \begin{pmatrix} \sin(x+2) + \sin(y+4) \\ \cos(x+2) + \cos(y+4) \end{pmatrix}.\end{aligned}$$

The velocity fields $\mathbf{u}_{2a}$ and $\mathbf{u}_{2b}$ are divergence-free, while $\mathbf{u}_{2c}$ is divergent. The field $\mathbf{u}_{2b}$ has sufficiently high frequency that it will not be well-resolved on any test; it behaves essentially as random numbers.

2.5.1 Random triangle and random square test

We conduct simulation tests for random triangles and random squares to verify continuity at edges and the div-free properties in interior areas. We run eight sets of tests, corresponding to the choice of (S)quare or (T)riangle, (Q)uadratic or (C)ubic boundary polynomial degree, and whether the net flux over the boundary is divergence-(F)ree or (D)ivergent. Triangle vertices are sampled randomly within the range $[-1, 1] \times [-1, 1]$. Squares have side lengths uniformly sampled within $[0.1, 1]$. Element boundary conditions are selected by sampling velocities at sample points randomly with $\|\mathbf{u}\|_\infty \leq 1$. For divergence-free tests, the divergent mode is projected from the sample points.

Shape type	Case label	$\ell_\infty(\text{err})$	$\ell_\infty(\text{div})$
Triangle	TQF	1.26×10^{-14}	1.45×10^{-7}
	TCF	4.06×10^{-14}	7.02×10^{-7}
	TQD	1.19×10^{-14}	2.84
	TCD	2.41×10^{-14}	1.07
Square	SQF	1.2×10^{-13}	2.82×10^{-7}
	SCF	9.93×10^{-14}	5.62×10^{-8}
	SQD	9.13×10^{-14}	1.23
	SCD	5.58×10^{-14}	0.0935

Table 2.7: This table shows the results of the numerical continuity tests. Here, “T” refers to triangles, “S” refers to squares; “Q” refers to quadratic polynomials and “C” refers to cubic polynomials; and “F” means the velocity field was divergence free and “D” means the velocity field was divergent. We observe continuity to roundoff error in all cases. Divergence is at the truncation error level for the central difference stencil used to evaluate the divergence error on the divergence-free cases.

To test continuity we sample 100 points on the edges of the triangle or square. We then evaluate the ℓ_∞ norm of the difference between the boundary polynomials and the interpolation routine evaluated at those points. The result is shown as $\ell_\infty(\text{err})$ in Table 2.7. In each case, the errors are at roundoff levels, verifying that the interpolation achieves the prescribed element boundary conditions. To verify the divergence-free property, we sample 100 points in the interior and approximate the divergence with finite differences using

$$\nabla \cdot \mathbf{u} \approx \frac{u(x + \epsilon, y) - u(x - \epsilon, y)}{2\epsilon} + \frac{v(x, y + \epsilon) - v(x, y - \epsilon)}{2\epsilon} \quad (2.14)$$

with $\epsilon = 10^{-5}$. We then compute the maximum absolute value of the divergence across the sample points. The result is shown as $\ell_\infty(\text{div})$ in Table 2.7. In the divergence-free cases, the maximum divergence is due to the accuracy of the finite difference approximation, though a bit higher than we might expect. We explore this error more in the following numerical test. In the divergent cases, the maximum divergence is something nonzero but finite, as would

be expected. Thus we see that the proposed interpolation scheme is analytically divergence free when the element boundary conditions have zero net flux, but it is always continuous.

2.5.2 Visualization of div-free errors in canonical objects

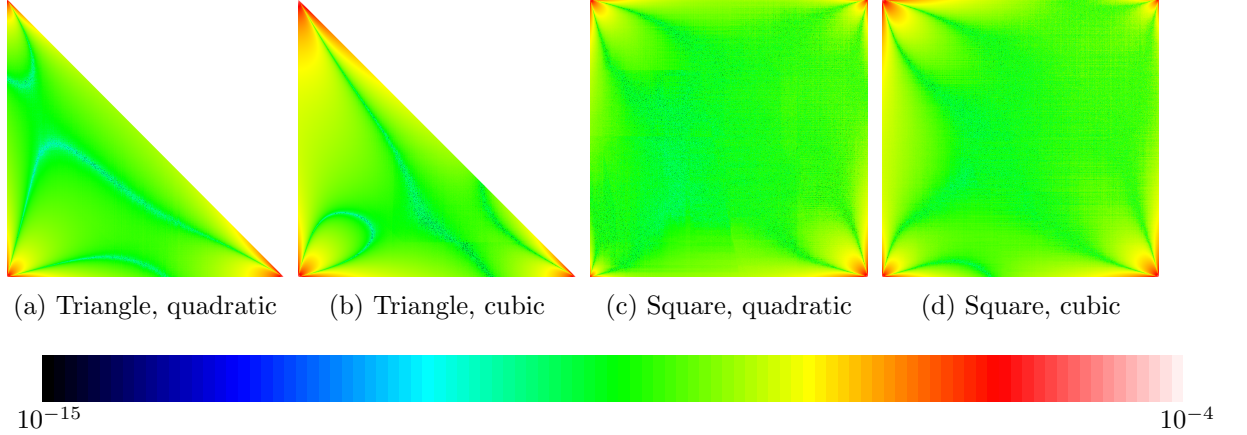


Figure 2.6: For each of the four combinations above, we constructed an interpolated velocity from random but flux-consistent sample values (in the sense that the total flux into the triangle is zero). Each of these velocity fields is analytically divergence-free everywhere. The plots show the central difference approximation to the divergence, colored using the color map shown. While the velocity field is analytically divergence-free everywhere, it is not C^1 at the corners. This causes the central difference approximation of the divergence errors to be much larger near the corners. (This figure is best viewed using the online version of this paper.)

In our next numerical test, we visualize the divergence of the interpolation. The setup is the same as in the previous test, except that we only consider the divergence-free cases and fix the triangle or square to be the canonical one. Velocities are initialized as before. For each pixel in the image, we approximate the divergence numerically using central differences as before. The results are shown in Figure 2.6. The divergence in color on a log scale. We observe that the computed errors are generally green away near the middle, which corresponds to an error on the order of 10^{-12} , which is consistent with our

finite differencing errors. The errors near the corners are much higher. This is because the finite difference approximation is less accurate near the corners. Although the divergence is analytically zero everywhere, the velocity gradient does not exist at the corners and has sharp gradients nearby. This degrades the accuracy of the finite difference approximation near corners.

2.5.3 Visualization of non-polynomial basis functions

In this section, we plot the non-polynomial basis functions for visualization purposes. For triangles, there are two rational basis functions for quadratic and three rational basis functions for cubic, shown in Figure 2.7. In each case, the basis function's value was normalized to the range $[0, 1]$, with the u component encoded in the red channel and the v component encoded in the green channel. The blue channel is zero. Observe that the color changes are smooth and well-behaved, consistent with the Lipschitz continuity of the basis functions. Equivalent plots for the square are shown in Figures 2.8. In the square case, there are four basis functions for both cubic and quadratic. As in the triangle case, the basis functions appear smooth, consistent with their Lipschitz continuity.

Unlike the velocity values, the velocity gradients of the rational function basis functions are not continuous at the corners. To observe this, we plot $\|\nabla \mathbf{u}\|_F$ for each of the rational bases as a color map, with the values mapped to the interval $[0, 1]$. The rational bases for triangles are shown in Figure 2.9. The three subfigures from the right show the three modes for the cubic case. Each of these basis functions fails to be C^1 at one of its vertices. Observe that the plots are smooth everywhere, except near one of the corners,

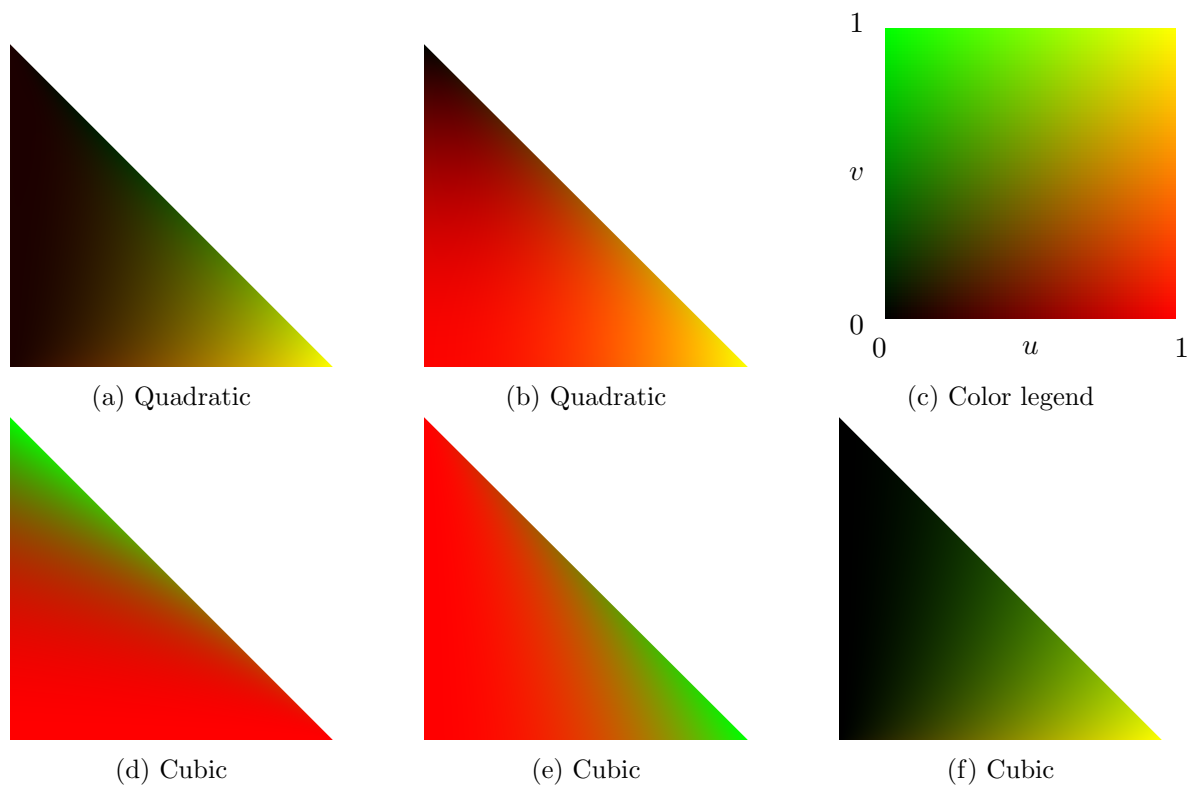
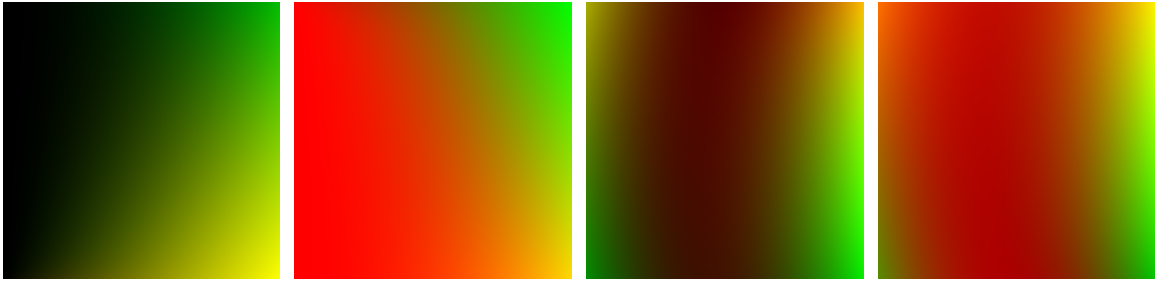
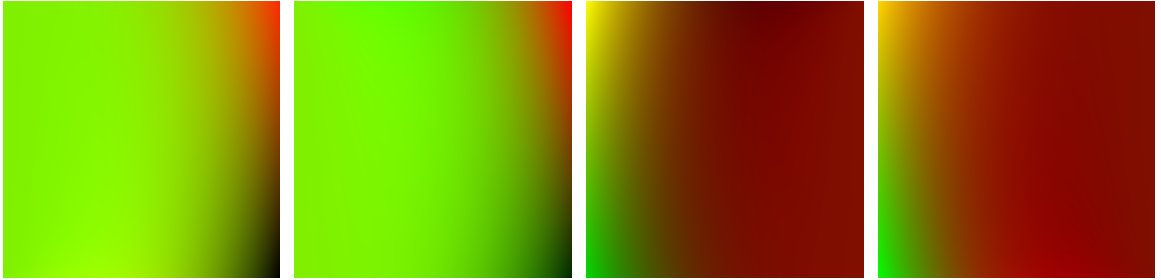


Figure 2.7: The figure shows the two quadratic and three cubic non-polynomial basis functions for the canonical triangle. Here u is encoded as red and v is encoded as green, with the ranges mapped into the interval $[0, 1]$, as shown in the color legend. Observe that the colors are smoothly varying, a consequence of the Lipschitz continuity of the interpolation scheme. (This figure is best viewed using the online version of this paper.)



(a) Rational modes for the **quadratic** square basis.



(b) Rational modes for the **cubic** square basis.

Figure 2.8: The figure shows the rational basis functions for the quadratic and cubic cases for the canonical square. Here u is encoded as red and v is encoded as green, with the ranges mapped into the interval $[0, 1]$. See Figure 2.7 for a color legend. Observe that the colors are smoothly varying, a consequence of the Lipschitz continuity of the interpolation scheme. (This figure is best viewed using the online version of this paper.)

where the velocity gradients exhibit high spacial frequency. The two plots for the quadratic case, shown on the left of the row, are similar, except that the high spatial frequency is observed at two vertices; this is because the modes are formed by combining two of the cubic rational basis modes. The square case is shown in Figures 2.10. Note that the corner issues are confined relatively closer to the vertices in the square case, and the frequency variations are somewhat less pronounced in the Frobenius norm.

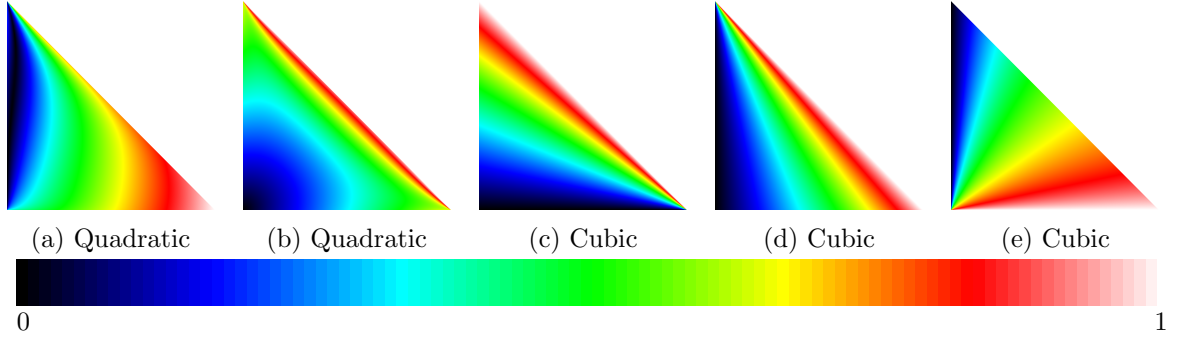


Figure 2.9: This figure shows the value of $\|\nabla \mathbf{u}\|_F$ for the rational basis functions for the canonical triangle. For each rational basis, the values are mapped to $[0, 1]$. There are two such basis functions for the quadratic case and three for the cubic case. In the cubic case, each triangle has sharp velocity gradients at one vertex and is well-behaved elsewhere. The quadratic modes are each constructed from two of the cubic modes and thus show sharp gradients at two vertices.

2.5.4 Refinement Study

In this section, we consider the convergence of the proposed divergence-free interpolation algorithm for MAC velocity fields on domains with irregular boundaries. For this test, we use a MAC grid on $[0, 1]^2$ cut into a ring shape by two concentric circles with radii 0.25 and 0.45 centered at $(0.5, 0.5)$, as illustrated in Figure 2.11. On this geometry, we consider three different velocity fields on the MAC grid. (1) The normal velocity components

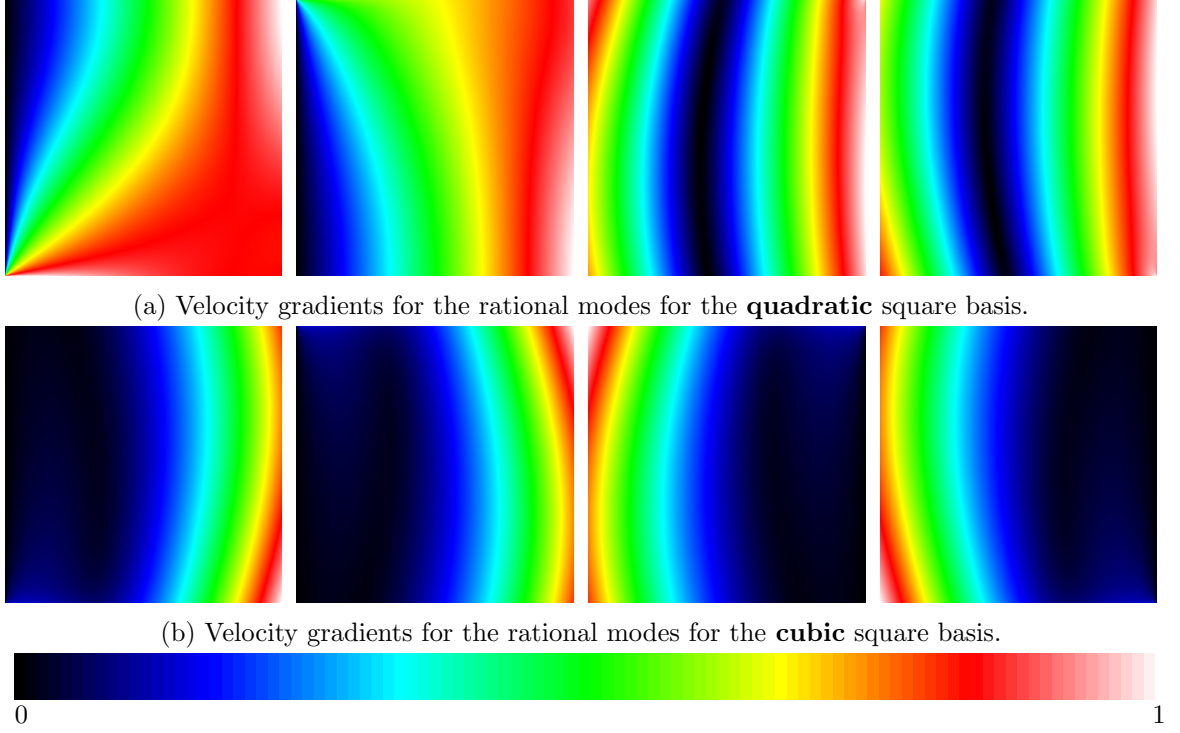


Figure 2.10: This figure shows the value of $\|\nabla \mathbf{u}\|_F$ for the rational basis functions for the canonical square. For each rational basis, the values are remapped to $[0, 1]$. In each basis function, the velocity gradients are well-behaved except at one vertex.

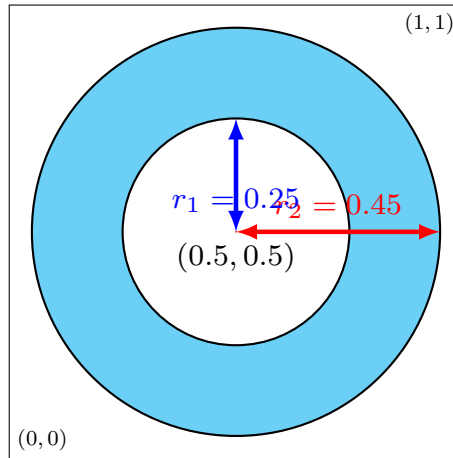


Figure 2.11: This figure shows the ring-shaped domain used in the numerical tests.

stored on the faces of the MAC grid are initialized from the divergence-free field $\mathbf{u}_{2a}$ by using the integrated average of the normal component of the analytic velocity $\mathbf{u}_{2a}$ over the faces of the MAC grid. Cut faces are integrated over the portion of the face that is inside the domain. The boundary condition imposed over boundary edges of the discretization is computed as the integrated average of the normal component of the domain boundary conditions over those edges. In this way, the MAC velocities are flux consistent; the flux across the face is equal to the velocity sample times the area of the face. Our interpolation is discretely divergence-free under these conditions. (2) The MAC grid data is initialized from $\mathbf{u}_{2a}$ by sampling it at the face centers. Our interpolation will be approximately, but not exactly, divergence-free under these conditions, since the MAC grid data will not be flux consistent. (3) The MAC grid data is initialized from the divergent field $\mathbf{u}_{2c}$ by sampling it at the face centers. Our interpolated result will be divergent on this field. We study the convergence under refinement by refining the MAC grid under all three scenarios.

To evaluate and compare the errors across resolution, we fix 10^6 randomly sampled points inside the domain (not at grid locations). The sample points are fixed across all resolutions, and we ensure that all of the sample points are inside the discretized domains at all resolutions. We estimate the divergence at these sample points using finite differences as in Section 2.5.1. The ℓ_2 and ℓ_∞ errors at these sample points are shown in Table 2.8. For the divergence-free analytic velocity field $\mathbf{u}_{2a}$, we observe second order in ℓ_2 . Errors in ℓ_∞ are much noisier, but the error generally stays within a small constant of the ℓ_2 error. The divergent case is significantly more noisy, under both norms, but the overall order is still approximately second order. We expect to see second order convergence, since our

$\mathbf{u}$	bc	Res	$\ell_2(\text{err})$	Order of error(ℓ_2)	$\ell_\infty(\text{err})$	Order of error(ℓ_∞)	$\ell_2(\text{div})$	Order of error(ℓ_2)	$\ell_\infty(\text{div})$	Order of error(ℓ_∞)
$\mathbf{u}_{2a}$	Ave	8	8.20×10^{-4}	-	3.67×10^{-3}	-	1.46×10^{-3}	-	3.90×10^{-2}	-
		16	2.35×10^{-4}	1.80	1.54×10^{-3}	1.25	9.10×10^{-6}	7.32	9.10×10^{-3}	2.10
		32	5.19×10^{-5}	2.18	7.03×10^{-4}	1.14	8.50×10^{-9}	10.06	7.44×10^{-7}	13.58
		64	1.23×10^{-5}	2.08	3.35×10^{-4}	1.07	7.50×10^{-9}	0.18	1.41×10^{-6}	-0.92
		128	2.98×10^{-6}	2.04	1.83×10^{-5}	4.19	2.05×10^{-7}	-4.77	2.05×10^{-4}	-7.18
		256	7.34×10^{-7}	2.02	3.79×10^{-6}	2.27	6.10×10^{-9}	5.07	3.05×10^{-7}	9.39
		512	1.82×10^{-7}	2.01	9.54×10^{-7}	1.99	5.90×10^{-9}	0.05	1.64×10^{-7}	0.90
	Sample	8	9.15×10^{-4}	-	4.21×10^{-3}	-	5.18×10^{-3}	-	4.90×10^{-2}	-
		16	2.56×10^{-4}	1.84	1.79×10^{-3}	1.23	1.85×10^{-3}	1.48	3.11×10^{-2}	0.65
		32	5.77×10^{-5}	2.15	5.95×10^{-4}	1.59	5.51×10^{-4}	1.75	3.16×10^{-2}	-0.02
		64	1.38×10^{-5}	2.06	2.84×10^{-4}	1.07	1.56×10^{-4}	1.82	6.35×10^{-3}	2.32
		128	3.39×10^{-6}	2.03	1.72×10^{-5}	4.04	4.88×10^{-5}	1.67	2.65×10^{-3}	1.26
		256	8.37×10^{-7}	2.02	3.24×10^{-6}	2.41	1.27×10^{-5}	1.95	6.09×10^{-4}	2.12
		512	2.08×10^{-7}	2.01	8.16×10^{-7}	1.99	2.55×10^{-6}	2.31	1.39×10^{-4}	2.13
$\mathbf{u}_{2c}$	Sample	8	1.91×10^{-3}	-	6.45×10^{-3}	-	2.35×10^{-1}	-	4.81×10^{-1}	-
		16	5.03×10^{-4}	1.93	2.36×10^{-3}	1.45	2.37×10^{-1}	-0.01	5.06×10^{-1}	-0.07
		32	1.16×10^{-4}	2.11	8.24×10^{-4}	1.52	2.38×10^{-1}	-0.00	5.19×10^{-1}	-0.04
		64	2.74×10^{-5}	2.08	3.57×10^{-4}	1.21	2.38×10^{-1}	-0.00	5.19×10^{-1}	-0.00
		128	6.65×10^{-6}	2.04	2.47×10^{-5}	3.85	2.38×10^{-1}	-0.00	5.21×10^{-1}	-0.00
		256	1.64×10^{-6}	2.02	3.89×10^{-6}	2.67	2.38×10^{-1}	-0.00	5.23×10^{-1}	-0.00
		512	4.05×10^{-7}	2.01	9.68×10^{-7}	2.01	2.38×10^{-1}	0.00	5.23×10^{-1}	-0.00
		1024	1.01×10^{-7}	2.01	2.40×10^{-7}	2.01	2.38×10^{-1}	0.00	5.23×10^{-1}	-0.00
		2048	2.51×10^{-8}	2.00	5.66×10^{-8}	2.08	2.38×10^{-1}	0.00	5.23×10^{-1}	0.00
		4096	6.30×10^{-9}	1.99	7.50×10^{-9}	2.92	2.38×10^{-1}	0.00	5.23×10^{-1}	0.00

Table 2.8: This table shows the interpolation error in a ring shaped domain with velocity field $\mathbf{u}_{2a}$ and $\mathbf{u}_{2c}$ using the **full** interpolation basis. $\mathbf{u}_{2a}$ field is divergence-free while the $\mathbf{u}_{2c}$ field is divergent. “Ave” means the grid data was populated from the analytic velocity by averaging over the face. “Sample” means the velocity was obtained by sampling the analytic velocity at the face center.

interpolation scheme recovers affine velocity fields, but it will not recover quadratic velocity fields.

The divergences are also instructive. For the flux-consistent data, divergences are generally low and reflect errors in the fixed central differencing stencil used to numerically approximate the divergence. Occasionally much larger errors are observed, which occurs when the central difference stencil crosses between elements. Since the interpolated velocity is analytically divergence-free, the central difference approximation of its divergence does not improve with refinement. In the sampled case, the interpolated velocity is not analytically divergence free; instead, the errors decrease with refinement as the accuracy of the reconstruction improves. In the case of the divergent velocity field, the divergence instead converges to its correct nonzero value under refinement.

Note on refinement strategies When performing refinement studies, care is required when deciding how the refinement should be performed. In the case of cut-cell strategies such as ours, the usual strategy is to simply refine the background grid, which reflects the way these methods are used in practice. Each level of refinement is a cut cell discretization, where the cut parts occupy a progressively smaller fraction of the full domain. This tends to produce noisy convergence, especially in ℓ_∞ , since the grid will be cut differently at each level of refinement.

2.5.5 Streamline of the div-free velocity field

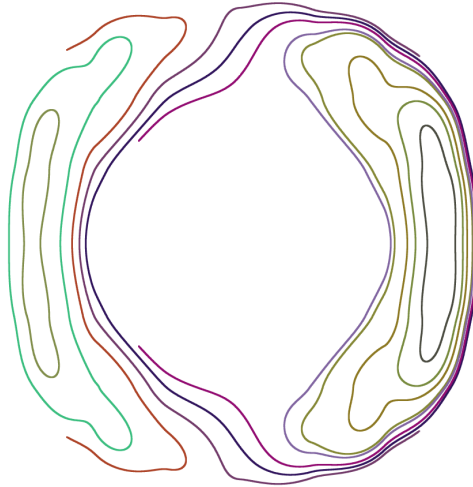


Figure 2.12: Divergence-free interpolation exhibits closed streamlines even in irregular flows through irregular domains at low resolution. Although the analytic velocity has zero normal component, the discrete boundary conditions are computed on a polygonal approximation of the circular boundaries and as such has nonzero normal components. This causes a few trajectories to exit the domain through the boundary.

In this section we visualize the streamlines of an interpolated divergence-free velocity field. Because our velocity field is divergence-free, these streamlines should form closed

loops or leave the domain. For this test, we use the ring geometry (see Figure 2.11) with the divergence-free velocity field

$$\mathbf{u} = \mathbf{u}_\alpha(x - \tfrac{1}{2}, y - \tfrac{1}{2}) + \mathbf{u}_\beta(x - \tfrac{1}{2}, y - \tfrac{1}{2}) \quad (2.15)$$

$$\mathbf{u}_\alpha(x, y) = (x^2 + y^2 + 1) \begin{pmatrix} -y \\ x \end{pmatrix} \quad (2.16)$$

$$\mathbf{u}_\beta(x, y) = \begin{pmatrix} 2xy(R^2 + r^2 - 2x^2 - 2y^2) \\ R^2r^2 - (3x^2 + y^2)(R^2 + r^2) + (5x^2 + y^2)(x^2 + y^2) \end{pmatrix} \quad (2.17)$$

where $r = .25$ and $R = .45$ are the inner and outer radii of the annulus. This velocity field has the property that it has zero normal velocity but nonzero tangential velocity along the boundaries of the annulus. It also contains flow around the annulus as well as local vortices. We use a low-resolution 8×8 grid so that the velocity field and geometry are poorly-resolved to make interpolation errors more visually obvious. We construct the streamlines as in [67]. The results are shown in Figure 2.12. As in [67], we observe that our trajectories are generally closed loops, both in the form of local vortices and flows around the annulus. We do not observe spirals, which would be observed if the interpolation were not divergence-free. We also observe three trajectories that reach the boundary. Although the analytical velocity does not contain a normal component along the (analytical) boundary, the same is not true of the discretized polygonal boundary, which does contain normal velocity components. As such, the interpolated velocity also contains small normal components, which allow streamlines to exit the domain.

2.5.6 Full basis vs polynomial div-free basis

u	bc	Res	$\ell_2(\text{err})$	Order of error(ℓ_2)	$\ell_\infty(\text{err})$	Order of error(ℓ_∞)	$\ell_2(\text{div})$	Order of error(ℓ_2)	$\ell_\infty(\text{div})$	Order of error(ℓ_∞)
u_{2a}	Ave	8	7.78×10^{-4}	-	3.54×10^{-3}	-	4.24×10^{-3}	-	1.63×10^{-2}	-
		16	2.36×10^{-4}	1.72	9.27×10^{-4}	1.93	4.70	-10.11	3.36×10^3	-17.66
		32	5.19×10^{-5}	2.18	3.76×10^{-4}	1.30	2.91×10^{-1}	4.01	2.78×10^2	3.60
		64	1.23×10^{-5}	2.08	1.77×10^{-4}	1.09	4.80×10^{-1}	-0.72	4.80×10^2	-0.79
		128	2.99×10^{-6}	2.04	1.48×10^{-5}	3.58	5.42×10^{-2}	3.15	4.43×10^1	3.44
		256	7.35×10^{-7}	2.02	3.65×10^{-6}	2.02	2.24×10^{-5}	11.24	5.37×10^{-4}	16.33
		512	1.82×10^{-7}	2.01	9.25×10^{-7}	1.98	5.86×10^{-6}	1.93	1.92×10^{-4}	1.49
	Sample	8	7.69×10^{-4}	-	3.01×10^{-3}	-	7.01×10^{-3}	-	2.23×10^{-2}	-
		16	2.44×10^{-4}	1.66	9.52×10^{-4}	1.66	4.51	-9.33	3.36×10^3	-17.20
		32	5.65×10^{-5}	2.11	3.34×10^{-4}	1.51	2.99×10^{-1}	3.92	2.85×10^2	3.56
		64	1.37×10^{-5}	2.05	1.51×10^{-4}	1.15	5.05×10^{-1}	-0.76	5.05×10^2	-0.83
		128	3.37×10^{-6}	2.02	1.26×10^{-5}	3.57	5.47×10^{-2}	3.21	4.43×10^1	3.51
		256	8.36×10^{-7}	2.01	3.11×10^{-6}	2.02	2.89×10^{-5}	10.89	1.01×10^{-3}	15.43
		512	2.08×10^{-7}	2.01	7.86×10^{-7}	1.99	6.81×10^{-6}	2.09	2.91×10^{-4}	1.79
u_{2c}	Sample	8	1.53×10^{-3}	-	3.62×10^{-3}	-	2.38×10^{-1}	-	4.82×10^{-1}	-
		16	4.58×10^{-4}	1.74	1.12×10^{-3}	1.69	1.41×10^1	-5.89	1.41×10^4	-14.83
		32	1.12×10^{-4}	2.04	5.40×10^{-4}	1.05	3.82×10^{-1}	5.20	2.98×10^2	5.56
		64	2.68×10^{-5}	2.06	1.87×10^{-4}	1.53	2.41×10^{-1}	0.67	3.40×10^1	3.13
		128	6.58×10^{-6}	2.03	1.56×10^{-5}	3.59	2.43×10^{-1}	-0.01	4.62×10^1	-0.44
		256	1.63×10^{-6}	2.02	3.90×10^{-6}	2.00	2.38×10^{-1}	0.03	5.23×10^{-1}	6.47
		512	4.05×10^{-7}	2.01	9.72×10^{-7}	2.01	2.38×10^{-1}	0.00	5.23×10^{-1}	-0.00

Table 2.9: This table shows the interpolation error in a ring shaped domain with velocity field **u_{2a}** and **u_{2c}** using **only polynomial** basis functions. The **u_{2a}** field is divergence-free while the **u_{2c}** field is divergent. “Ave” means the grid data was populated from the analytic velocity by averaging over the face. “Sample” means the velocity was obtained by sampling the analytic velocity at the face center.

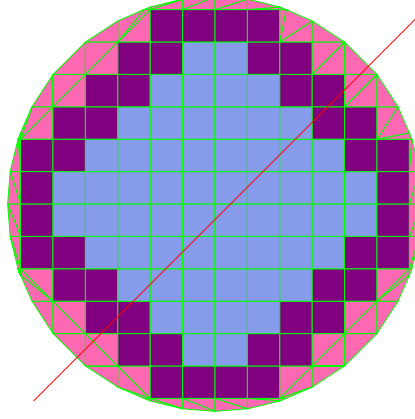
Our approach to constructing divergence-free velocity field on squares and triangles is based on using a maximal set of divergence-free polynomial basis modes and then supplementing them with rational functions to achieve a complete basis for the divergence-free subspace. An alternative strategy, such as when using Raviart-Thomas elements, is to instead just use polynomials. We compare against this alternative strategy within our basis framework by omitting the rational basis functions to see how this affects the behavior. Since we no longer have a full basis, the matrix **K** that we invert will no longer be full rank, so we use its pseudoinverse instead. This modified scheme effectively chooses the polynomial divergence-free interpolation that most accurately fits the element boundary conditions. Since the interpolation does not match its specified boundary velocities, we expect that the interpolation will be discontinuous across elements. But otherwise, we expect the scheme

to be analytically divergence free and second order accurate under the same conditions as when the rational basis functions are included. To evaluate the accuracy, we repeat the refinement study Section 2.5.4 with the rational basis functions omitted. The results are shown in Table 2.9. The overall interpolation accuracy is comparable, which suggests that the incomplete basis is nonetheless sufficient for accurate interpolation. In the case of the divergences, however, we observe that very large divergence estimates are observed in some cases. This is an artifact that occurs when a finite difference stencil crossing a discontinuity. Because the interpolation converges under refinement, the magnitude of the discontinuities should decrease with refinement as well. For applications where continuity is not essential, omitting the rational basis functions may be entirely reasonable.

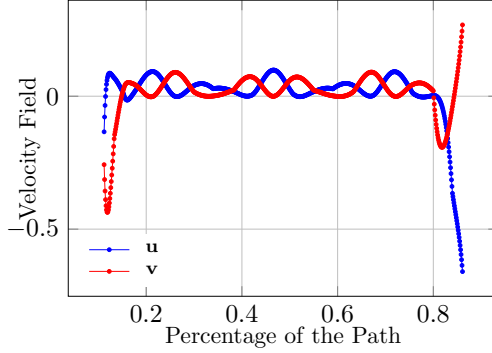
2.5.7 Path trace plotting

In Section 2.5.1, we explored the interpolation continuity properties of individual triangles or squares. In this section, we wish to explore the continuity for the interpolation over cut MAC grids and also the role that the rational modes play in achieving continuity between elements.

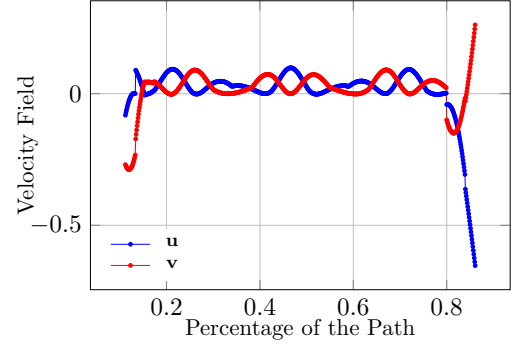
For this test, we use a circular domain on a low-resolution 16×16 grid. The discretization of this domain is shown in Figure 2.13, with colors differentiating triangulated cut cells, transition cells, and interior cells. We initialize the MAC grid from the high-frequency $\mathbf{u}_{2b}$ using face-integrated values to obtain flux-consistent data on the MAC grid. Since this velocity field is not resolved at this resolution, the MAC data behaves like random



(a) Cell classification



(b) Full interpolation basis



(c) Polynomial-only basis

Figure 2.13: The top illustration shows the different types of cells within a circular domain. The red diagonal line shows a path that crosses the interior region. This path visits all cell types as well as all types of transitions that can occur between types. The input velocity field of the domain is $\mathbf{u}_{2b}$, which is divergence-free but sufficiently high frequency that it cannot be resolved on this grid. The bottom-left plot shows the interpolated velocity along the red path when interpolating using the full bases. The red line indicates the u component, and the blue line indicates the v component. The interpolated velocity is continuous along the entire path. The bottom-right plot shows the interpolated velocity of the same path using only polynomial bases. This path contains discontinuities as the path transitions from one element to the next. The use of an unresolved data causes some of these discontinuities to be large enough to be visually observed. When interpolating smooth data, these discontinuities would be too small to visually observe.

but flux consistent data, which tends to make undesirable numerical properties visually apparent.

We examine the interpolated velocity profile along a path through the domain. This path was chosen so that it crosses all types of transitions. The path being traversed and the results are shown in Figure 2.13. We observe that the interpolated values along the path are C^0 continuous in both u and v components but appear to have slope discontinuities. This is the level of continuity that we expected.

Polynomial basis only We repeat this path continuity test with the rational basis functions omitted, as described in Section 2.5.6. The resulting trace is also shown in Figure 2.13. Compared with the full scheme, we observe discontinuities in the trace. Without a full basis, the interpolated field is not able to match the prescribed velocity profile along the boundary; only a least-squares fit is achieved. Because of this, the velocity profiles along the boundary for adjacent elements need not match up at the boundary, which results in discontinuities. Although omitting the rational modes leads to discontinuities, these may be acceptable if the application is not sensitive to them. Since the interpolation accuracy is second order, the interpolation errors will be small if the field is resolved, so that any discontinuities must also be small.

2.5.8 Random velocity field continuity test

In this section, we test the compatibility of our full cut-cell interpolation scheme on an irregular domain. For this test, we use a $[0 - 1]^2$ MAC grid with 32^2 resolution. We cut the regular grid with a circle centered at $(\frac{1}{2}, \frac{1}{2})$ with radius 0.3; the interior of the

circle is our domain. We initialize the MAC faces with uniform random data in the range $[-1, 1]$ and set boundary velocity to zero to ensure compatibility. We then project this field with cut-cell projection discretization of Ng et al. [63] to produce a discretely divergence-free MAC vector field. We note that the resulting velocity field has high frequency in the interior (since the field can vary from -1 to 1 over a single grid cell). The reconstruction also has high shear at the boundary, since the projection of Ng et al. [63] naturally enforces slipping boundary conditions (it never enforces zero tangential velocity) while our boundary condition for this test is zero in the tangential direction as well.

To evaluate the effectiveness of the cut-cell interpolation, we sample the circle's interior with 10^6 points, excluding a narrow band of width 10^{-3} to ensure the sample points lie inside the discretized polygonal region. We approximate the divergence at each sample point using (2.14) with $\epsilon = 10^{-8}$. We note that we have used an ϵ small enough to avoid approximating divergence using velocity samples from different elements. The reconstruction is only C^0 , and mixing samples between elements leads to large errors in the finite difference approximation of divergence. In this test, we found the ℓ_2 and ℓ_∞ errors in the central-difference approximation of the divergence were 1.32×10^{-7} and 2.56×10^{-5} , which is consistent with what was observed when testing on individual elements.

We evaluate continuity by approximating the Lipschitz bound $L(\mathbf{u}, \Omega)$ at each sample point. To do this, we randomly select $\Delta \mathbf{x} \in [-\delta, \delta]^2$ with $\delta = 10^{-3}$ at each sample point, then use (2.10) with $\mathbf{y} = \mathbf{x} + \Delta \mathbf{x}$. In this test, sample points are allowed to cross elements, which ensures continuity between elements. The ℓ_2 and ℓ_∞ norms of the resulting bounds are 5.56×10^1 and 3.52×10^3 . Note that the ℓ_2 bound is quite close to the expected bound for the input, which we would expect to be approximately $2/\Delta x = 64$. About 0.13%

of sample points exceed this by a factor of ten, with the worst sample point exceeding it by a factor of almost 60. In practice, these sample points are observed to occur near triangle edges where the reconstructed velocity profile has strong tangential shear. This suggests that a more sophisticated selection of tangential velocities along boundaries between elements may significantly reduce smoothness in velocity gradients.

We investigate the continuity and divergence of the interpolated velocity field derived from a random velocity field. In this context, a random velocity field indicates that the input MAC grid velocities are randomly sampled. We employ Ng et al. [63] to project this velocity field into a div-free velocity field, followed by interpolation based on that projection. Our testing methodology is as follows:

- The domain is defined as a circle. We project the random velocity field and initialize both div-free squares and triangles, with the input boundary velocity and boundary average velocity function set to 0. We denote the interpolated function as U .
- We randomly sample 10^6 points within the domain, ensuring they are positioned a small ϵ distance away from the boundary to confirm they lie inside the domain.
- The discrete divergence at each point is calculated using equation 2.14, with Δt set to 10^{-8} .
- The Lipschitz constant is computed locally for each point, $\mathbf{x}$. We randomly sample $\Delta \mathbf{x}$ from the range $[-10^{-3}, 10^{-3}] \times [-10^{-3}, 10^{-3}]$. The Lipschitz constant for $\mathbf{x}$ is then given by

$$L_{\mathbf{x}} = \frac{\|U(\mathbf{x} + \Delta \mathbf{x}) - U(\mathbf{x})\|}{\|\Delta \mathbf{x}\|} \quad (2.18)$$

- Finally, we calculate both the ℓ_2 norm and the ℓ_∞ norm of the divergence and $L_{\mathbf{x}}$ across all 10^6 points.

We get the ℓ_2 norm of the $L_{\mathbf{x}}$ is 5.56×10^1 . ℓ_∞ norm of the $L_{\mathbf{x}}$ is 3.52×10^3 , and the ℓ_2 norm of the divergence is 1.32×10^{-7} and the ℓ_∞ norm of the divergence is 2.59×10^{-5} . From the results, we could tell that the interpolated velocity field is divergence-free. Additionally, we successfully project a random velocity field into a div-free field, as evidenced by the $\ell_2(\text{div})$ norm of 10^{-7} and the $\ell_\infty(\text{div})$ norm of 10^{-5} .

2.5.9 Analytical velocity field projection test

In this section, we construct a divergent velocity field

$$\mathbf{u}' = \mathbf{u}_{2a} + \begin{pmatrix} (x^2 - x)(y^3/3 - y^2/2) \\ (y^2 - y)(x^3/3 - x^2/2) \end{pmatrix}, \quad (2.19)$$

and project this field to be discretely divergence-free using the projection scheme of Ng et al. [63] to demonstrate compatibility with the projection scheme. The projection scheme solves a discrete Poisson problem for the pressure and uses it to project the velocity field onto a divergence-free vector field on an irregular domain. When a velocity field is decomposed into a divergence-free part and a curl-free part, the projection scheme should recover the divergence-free component. In this case, the divergence-free component is $\mathbf{u}_{2a}$, so the projected velocity field should match $\mathbf{u}_{2a}$. We use the irregular domain shown in Figure 2.11. We initialize the MAC grid velocity by sampling the divergent velocity field $\mathbf{u}'$ at the face centers. Cut faces are sampled at the midpoint of the inside portion of the face. Note that we do not initialize the MAC data by integrating over the faces for this

test; the discretely divergence-free velocity field is obtained through projection by solving the Poisson equation. We denote the projected velocity as $\mathbf{w}$, so that $\mathbf{w} - \mathbf{u}_{2a}$ is the error. We sampled 10^6 points within the domain and use the equation (2.14) with $\epsilon = 10^{-8}$ to approximate the divergence. We run this test with resolutions from 32 to 1024; the results are shown in Table 2.10. We observe roughly second error convergence, which is what we expect for Ng et al. [63]. The divergence is not significantly affected by resolution, since the interpolated velocity is analytically divergence free and the divergence stencil is not changing (ϵ is fixed).

$\mathbf{u}$	bc	Res	$\ell_2(\text{div})$	Order of error(ℓ_2)	$\ell_\infty(\text{div})$	Order of error(ℓ_∞)	$\ell_2(\text{err})$	Order of error(ℓ_2)	$\ell_\infty(\text{err})$	Order of error(ℓ_∞)
$\mathbf{u}_{2a}$	Sample	32	3.54×10^{-6}	-	3.54×10^{-3}	-	9.01×10^{-5}	-	2.04×10^{-3}	-
		64	7.10×10^{-9}	8.96	7.88×10^{-7}	12.13	2.56×10^{-5}	1.81	6.85×10^{-4}	1.58
		128	6.40×10^{-9}	0.15	3.61×10^{-7}	1.13	7.49×10^{-6}	1.77	2.73×10^{-4}	1.33
		256	6.10×10^{-9}	0.07	2.36×10^{-7}	0.61	2.03×10^{-6}	1.88	8.87×10^{-5}	1.62
		512	6.00×10^{-9}	0.02	3.05×10^{-7}	-0.37	4.32×10^{-7}	2.23	2.45×10^{-5}	1.85
		1024	5.90×10^{-9}	0.02	3.33×10^{-8}	3.20	7.50×10^{-8}	2.53	4.20×10^{-6}	2.55

Table 2.10: This table shows the results of projecting a divergence velocity field using the projection scheme from [63]. Interpolating this projected data yields a velocity field that is analytically divergence-free. The divergence errors are due to the truncation error in the central differencing divergence stencil, which is not changing under refinement. The errors show the difference between the interpolated projected velocity and the analytical projected result. The convergence order is approximately two.

Although the divergence-free interpolation might seem very expensive, we did not find this to be the case, especially compared with other computational costs that would be involved in applications. On this test at the highest resolution (1024), we observed that applying the projection from Ng et al. [63] was by the far the most expensive step, taking 6.95 s; an equivalent step will almost always be included in any computational scheme that produces divergence-free grid data. By contrast, precomputing the divergence-free squares and triangles took 0.03 s, and performing the evaluation for the test (which included 6

million div-free interpolations) took 0.40 *s*. Other miscellaneous overhead on the test case took 0.26 *s*.

2.6 Conclusion

In this paper, we have presented analytically divergence-free interpolation schemes over triangles and squares, where the interpolated vector fields at the boundary are arbitrary quadratic or cubic polynomials. We then show that these discretizations can be used to extend an existing divergence-free interpolation scheme designed for regular MAC grids to handle irregular boundaries. The resulting scheme is analytically divergence-free and C^0 continuous everywhere. This allows efficient schemes designed for regular grids to be used in the interior while also allowing an analytically divergence-free vector field to be constructed up to the boundary. Although the schemes for triangles and squares are not differentiable at the corners, we show that they are nevertheless Lipschitz continuous, provided the triangles themselves are well-conditioned.

The scheme presented is specific to 2D, since we present discretizations for triangles and squares but not 3D shapes like cubes or tetrahedra. The discretizations are readily extended to higher degree polynomials at the boundary, since polynomials suffice for all additional basis functions. The algorithm employed for discretizing the portion of the grid near the boundary does not attempt to ensure well-conditioned triangles, nor does it attempt to ensure Lipschitz continuity within those triangles. The general strategy relies on the interpolation scheme used for the interior being flux-consistent, which limits the range of schemes that can be used in the interior. We leave these considerations for future work.

Chapter 3

Divergence-free Interpolation of Three-dimensional MAC Grid Data over Irregular Domains

3.1 Introduction

Divergence-free vector fields appear widely in computational physics and form the foundation for numerical methods for many conservation laws. In magnetohydrodynamics, the solenoidal constraint on the magnetic field stems from Maxwell's equations, and preserving it discretely is essential for stable, physically meaningful simulations of magnetized fluids [9, 29, 89, 4, 5, 6, 8]. The same constraint governs the velocity field in incompressible flow, making divergence-free representations a recurring requirement for the incompressible Navier-Stokes [41, 10, 66, 55, 83] and Stokes equations [49, 45, 48].

A common approach in such simulations is to store fields on a structured grid and enforce the divergence constraint globally, often through a projection step that produces a discretely divergence-free field. This is adequate when the field is evaluated only at grid locations. Many algorithms, however, require values at arbitrary points: semi-Lagrangian methods trace characteristics through cells, particle methods move samples continuously, and visualization tools query fields along streamlines or other geometric paths. Standard interpolants are efficient and well understood, but they generally do not preserve the divergence constraint satisfied by the discrete data. This gap has motivated reconstruction methods whose outputs are analytically divergence-free.

The need for such interpolants is especially clear in particle tracking. Spurious divergence introduced by standard interpolation can produce artificial particle clustering or depletion in regions with strong velocity gradients. Jenny et al. [41] introduce a conservative velocity interpolation scheme for tracking particles in a finite-volume flow solver; this idea was shown by Meyer and Jenny [58] to largely eliminate spurious distribution of particles in two-dimensional incompressible flow and was later extended to three dimensions by Wang et al. [85], where it was shown to improve particle distributions in geodynamic flows with large viscosity contrasts. Beyond particle tracking, the same need arises wherever discrete divergence-free fields must be evaluated away from grid locations, and several families of analytically divergence-free interpolants have been developed to meet it.

One family of approaches constructs the divergence-free interpolant globally. Radial basis function methods accommodate scattered or unstructured samples in arbitrary dimensions through a global solve over matrix-valued divergence-free kernels [61, 54, 57, 30], with related global expansions in orthogonal bases for three-dimensional magnetic field in-

terpolation [89]. Drake et al. [28] avoid the global system by combining local approximants built on separate sample subsets. In practice, global solves and boundary handling can limit the applicability of these methods in three-dimensional grid-based fluid settings [20]. Wavelet methods instead use a multi-scale basis on structured grids, well-suited to problems with rich multi-resolution content [16, 82, 26], including extensions that incorporate different boundary conditions [77, 43, 78]. These methods address challenges different from ours, such as unstructured samples or multiscale fields, and are not tailored to the structured, locally sampled setting considered here, where global approaches can be more expensive than local reconstruction schemes.

Another major class of methods constructs a vector potential $\mathbf{A}$ and represents the divergence-free field as $\nabla \times \mathbf{A}$, which automatically satisfies the divergence-free constraint [31, 66, 10, 39]. In 2D, the potential reduces to a scalar stream function, whereas in 3D it becomes a vector potential with gauge freedom; recovering a smooth, velocity-consistent potential typically requires a global solve. Finn and Chacón [31] use a tricubic spline vector potential for volume-preserving integration of solenoidal fields, and Ravu et al. [66] fit the derivatives of the B-spline vector potential directly to 3D grid velocities, both requiring the solution of a large global system. A different strategy first recovers discrete $\mathbf{A}$ values and then interpolates these values to obtain a continuous potential [10, 72, 20]. Bao et al. [10] first introduce this strategy, solving a vector Poisson problem for the discrete potential. On a periodic rectangular domain, the problem decouples into three scalar Poisson solves handled by FFT. Silberman et al. [72] and Chang et al. [20] replace the vector Poisson solve with a sweeping strategy that produces a candidate potential locally, followed by a single scalar Poisson solve to enforce a Coulomb gauge for smoothness. Silberman et al. [72]

still apply only to uniform grids, while Chang et al. [20] further support irregular cut-cell obstacles.

On regular grids, divergence-free interpolation is typically built within a polynomial space, with the constraint enforced by the choice of basis functions. Handscomb [37] gives one of the earliest such constructions, recovering velocity components as tensor-product splines from face-integrated flux data through a global solve. Most subsequent work has favored local reconstructions: Balsara [4] fits cell-wise quadratic polynomials on adaptively refined Cartesian blocks using face-centered flux constraints; this was later raised to third and fourth order using WENO to keep face moments non-oscillatory across the steep gradients of MHD shocks [6]. From a different angle, Jenny et al. [41] introduce a quasi-conservative two-dimensional velocity interpolation scheme designed to preserve accurate dilatation behavior for particle tracking, extended to three dimensions by Wang et al. [85]. Tóth and Roe [81] construct prolongation and restriction operators that preserve the discrete divergence across refinement levels. Schroeder et al. [70] give a dimension-independent recipe for local divergence-free polynomial interpolants with C^0 or C^1 continuity, and Roy-Chowdhury et al. [67] extend it to fourth order, add a curl-free variant, and introduce a flux-consistent formulation that we adopt in our scheme. These methods assume the simulation domain is grid-aligned and do not address how to enforce divergence-free constraint at the irregular cut-cell interfaces.

Beyond regular grids, divergence-free reconstructions have also been developed on polygonal or polyhedral elements in the finite element, virtual element, and finite volume literature [93, 12, 21]. These methods typically construct divergence-free polynomial bases on shape-regular polytopes, often based on Raviart-Thomas spaces or related $H(\text{div})$ -

conforming spaces [33, 86, 50]. On tetrahedral meshes, Balsara [5] develops a second-order divergence-free reconstruction. Balsara and Dumbser [7] extend this up to fifth order using a constrained L^2 projection to map a standard WENO reconstruction onto the divergence-free polynomial subspace. In both constructions, the reconstruction is normal-continuous across faces but tangentially discontinuous. Such approaches rely on meshes that conform to the domain geometry. In cut-cell methods, where a background grid is intersected by the domain boundary, element shapes are determined by intersection rather than mesh generation, and polynomials alone cannot form a complete divergence-free basis.

As an alternative to the methods above, cut-cell methods use a regular grid in the interior for efficiency and introduce cut cells near irregular boundaries. However, extending a divergence-free interpolant from the background grid to the cut region is nontrivial and has motivated substantial work.

Immersed boundary methods (IBMs) are one setting in which this problem arises. They embed an irregular boundary, such as a rigid body [59], a deformable body [64], or an interface [71, 91], in a background grid that does not conform to it. Some IBMs represent the boundary through a force distributed over nearby grid cells, while others apply special treatments in or near the cut cells produced by the boundary. This flexibility has made IBMs a powerful tool for flow simulations in complex geometries [84]; with applications across many practical fields [65, 69, 2]. Our method fits within this paradigm, using a regular background grid with special treatment near or within cut cells.

Several recent approaches have used this setting. Albuquerque et al. [1] construct a divergence-free least-squares interpolant on unstructured polyhedral grids in 2D. The interpolant is anchored at the body surface and does not require cut-cell shape informa-

tion. Kingora and Sadat-Hosseini [46] work in 3D with cut cells reshaped to conform to the immersed boundary, absorbing cut volumes and face areas into area-fraction-weighted coefficients. Thus, the discretization stencil remains uncut. Cut finite element methods (CutFEM) take a different approach. They retain a non-boundary-fitted background mesh, with modifications to the discrete spaces or stabilization terms near the interface. Frachon et al. [32] develop a cut finite element method for a Darcy interface problem using Raviart-Thomas or Brezzi-Douglas-Marini velocity spaces. The cut cells are not reconstructed. Instead, standard basis functions are retained on the whole background elements, and the cut geometry enters only through integration over the portions of each element lying in the physical subdomains. Liu et al. [53] construct an analogous CutFEM discretization for the Stokes problem using the Scott-Vogelius pair. The pair is defined on a macro-refined mesh, obtained by applying a Clough-Tocher split in 2D or an Alfeld split in 3D. The ghost penalty perturbs the discrete continuity equation, so the resulting velocity is pointwise divergence-free only outside an $O(h)$ neighborhood of the boundary. Chang et al. [20] take a different route. Instead of constructing a CutFEM space, they enforce boundary-respecting incompressibility by correcting a Cartesian vector-potential interpolant near the solid boundary via a ramping procedure. Their method avoids treating cut cells directly and keeps the fluid discretization on the background grid, using triangulation only to represent the solid surface. In contrast, our method reconstructs basis functions directly on the cut cell and solves for their coefficients locally.

Polynomial bases are the natural starting point for such reconstructions and are widely used because they are easy to work with and have good numerical properties. However, the space of divergence-free polynomials over simplices or tensor-product cells does

not form a complete basis in either two or three dimensions. The issue is more pronounced in three dimensions, where matching across a shared face requires representing arbitrary polynomials on a two-dimensional surface, and the divergence-free polynomial space inside the element does not supply enough degrees of freedom to span the required facial polynomials. Completing the basis requires including basis functions that are not polynomials. Guzmán and Neilan [34] construct a continuous divergence-free interpolation over tetrahedra in three dimensions by supplementing the usual polynomial basis with rational face and edge bubble functions, extending an earlier triangular construction Guzmán and Neilan [35] from two to three dimensions; closely related rational-bubble constructions appear in their concurrent work on plane elasticity [36]. Their construction is body-fitted: it assumes a conforming tetrahedral mesh of the domain and builds basis functions on whole simplices, leaving the case of cut cells on a background grid unaddressed. Working in the cut-cell setting, we propose rational functions to complete our basis for the tetrahedra and an analogous rational completion for cubes, extended from the 2D work [44]. They also construct a method for assembling these elements into a global interpolant for the irregular domain. We use spline-based divergence-free interpolation on MAC grids in the deep interior, where the stencil has access to enough surrounding data. Near the domain boundary, the spline-based reconstruction is no longer valid, leaving a gap between the deep interior and the boundary. We fill this gap with cubes and tetrahedra that carry their own divergence-free bases. The result is a continuous, divergence-free interpolant that extends all the way to the domain boundary.

3.2 Discretization

This paper extends the work of divergence-free interpolation on 2D irregular domains to 3D. As in 2D, we wish to perform analytically divergence-free interpolation from MAC grid data. Away from the boundary, we use an efficient spline-based flux-preserving divergence-free scheme [67]. We assume that the MAC grid data has been projected to be discretely divergence-free in the same sense as in 2D, namely that the flux across each MAC grid face is the velocity stored at that face times the area of that face that is inside the domain, and that the total flux within each MAC cell is zero. This is consistent with Ng et al. [63]. We assume that the velocity values along the boundary are known analytically; we refer to these as *domain boundary conditions*.

In the 2D case, once cut points are computed along faces, the fraction of the face that is inside is well-defined. In 3D, this is not the case. It is important that the face fraction that is considered to be inside for a cut MAC grid face is consistent with the MAC grid projection. In both cases, we compute this fraction using marching cubes. This will ensure consistency with our interpolation in cut cells.

As in 2D, the spline-based interpolation that we use in the interior cannot be used near boundaries, leaving a gap. We fill this gap using cubes and tetrahedra, which we refer to as *shapes*. As in 2D, we will compute polynomial boundary conditions for these shapes, which we refer to as *element boundary conditions*. This will ensure C^0 continuity across cell faces and between shapes.

3.2.1 Canonical shapes

Analogous with the 2D case, we define two canonical shapes: a cube and a tetrahedron. We can construct a divergence-free basis for these canonical shapes and then push the resulting velocity field forward to the original shape, which preserves the divergence-free property [44]. This allows the basis to be constructed in advance and hard-coded.

Tetrahedron

We use the tetrahedron with vertices at $(0,0,0)$, $(1,0,0)$, $(0,1,0)$, and $(0,0,1)$ as our canonical tetrahedron. In the case of the tetrahedron, we will need to be able to handle degree four polynomials as boundary conditions. This requirement comes as the result of needing continuity between tetrahedra and cubes; see Section 3.4. Since there are 15 monomials of the form $x^a y^b$ where $0 \leq a, b$ and $a + b \leq 4$, we will need 15 sample points on each face. This leads to the sample point layout with a sample points at each vertex, three sample points in the interior of each edge, and three sample points in the middle of the face, as shown in Figure 3.1. This leads to 34 sample points for the full tetrahedron. Since each sample point is a velocity sample that has three degrees of freedom in 3D, this leads to a total of 102 degrees of freedom in our sample points. We must therefore construct a basis with 102 functions, of which 101 will be divergence-free and one will be divergent. While this seems daunting, the task is manageable because most of these can simply be polynomials.

To construct our basis, we begin by selecting as many polynomials as possible. These basis functions are of the form $\bar{\mathbf{u}}_k = \langle p_{kx}(x, y, z), p_{ky}(x, y, z), p_{kz}(x, y, z) \rangle$ where

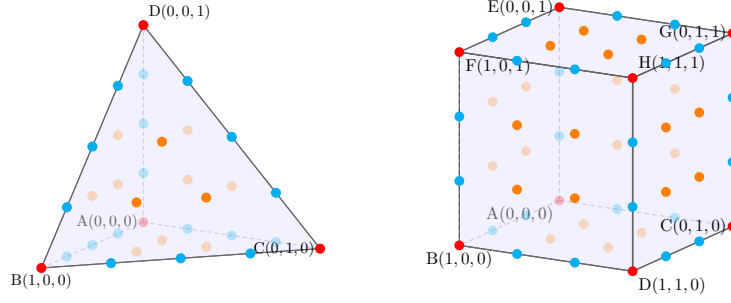


Figure 3.1: The left figure shows the canonical tetrahedron with vertices at $(0, 0, 0)$, $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$. The right figure shows the canonical cube with vertices at $(0, 0, 0)$, $(1, 0, 0)$, $(0, 1, 0)$, $(1, 1, 0)$, $(0, 0, 1)$, $(1, 0, 1)$, $(0, 1, 1)$ and $(1, 1, 1)$. Vertex samples (●), edge samples (●), and face samples (●) are indicated.

p_1, p_2, p_3 are polynomials and $\nabla \cdot \mathbf{u}_k = 0$. A basis vector is constructed from each basis function by evaluating it at the locations of the sample points and stacking the results into a vector of size 102. Basis functions are considered to be independent if their corresponding basis vectors are linearly independent. It turns out that 85 of the divergence-free basis functions as well as the divergent basis function can be polynomials. This leaves 16 modes that cannot be polynomials. As in the 2D case, it turns out that it suffices to construct these as rational functions.

Much of the difficulty in constructing a rational basis consists of identifying the form for the denominator polynomials. In 2D, the necessary rational functions had denominators consisting of polynomials of the form: $1 - x$, $1 - y$, and $x + y$. In the canonical triangle, $\lambda_1 = x$, $\lambda_2 = y$, and $\lambda_3 = 1 - x - y$ are the barycentric coordinates, which allows us to express these denominators as $\lambda_i + \lambda_j$. This was consistent with the form of the bubble functions from Guzmán and Neilan [36]; the two papers had basis functions that spanned the same space, though they differed in form. In 3D, the situation is much more complex. In the case of our canonical tetrahedron, our barycentric coordinates are

$\lambda_1 = x$, $\lambda_2 = y$, $\lambda_3 = z$, and $\lambda_4 = 1 - x - y - z$. Guzmán and Neilan [34] presented a 3D extension of the 2D bubble functions. They used rational functions with denominators like $(\lambda_i + \lambda_j)(\lambda_i + \lambda_k)(\lambda_j + \lambda_k)$, which they refer to as rational face bubble functions. They also included rational edge bubble functions, which have denominators of the form $(\lambda_i \lambda_j + b(\lambda_i + \lambda_j))(\lambda_i + \lambda_j)$.

We note that it is possible to form a full basis using only denominators of the form $(\lambda_i + \lambda_j)(\lambda_i + \lambda_k)(\lambda_j + \lambda_k)$. Denominators such as those in the rational edge bubble functions are unnecessary. Once the denominators are known, we only need to search the space of polynomials of some maximum degree for the numerators; this leads to a linear algebra problem. In particular, once we have fixed a denominator $d(x, y, z)$, we look for polynomials $p_k(x, y, z)$, where

$$\bar{\mathbf{u}}_k = \left\langle \frac{p_1(x, y, z)}{d(x, y, z)^2}, \frac{p_2(x, y, z)}{d(x, y, z)^2}, \frac{p_3(x, y, z)}{d(x, y, z)^2} \right\rangle. \quad (3.1)$$

Note that we allow the denominator to be squared, since we would like to allow $d(x, y, z)$ to occur in the denominator of the stream function. Simpler denominators are included in this description, since factors from the denominator can cancel with factors from the numerator. The resulting velocity field must be divergence free ($\nabla \cdot \bar{\mathbf{u}}_k = 0$). It must be a polynomial at the boundary, so that $\lim_{x \rightarrow 0} \bar{\mathbf{u}}_k$ is a vector of polynomials in y, z of total degree at most 4 (and similarly for $y \rightarrow 0$, $z \rightarrow 0$, and $z \rightarrow 1 - x - y$). As in prior methods, we require the rational functions to be continuous over the entire canonical shape, including the boundary. Our final tetrahedron basis is shown in Tables 3.5 and 3.6.

Cube

We use the unit cube $[0, 1]^3$ as our canonical cube. In the case of the square, we required the ability to handle cubic polynomials as boundary conditions, since the spline-based interpolation used in the interior uses polynomials of this degree. In 3D, the spline-based interpolation can produce polynomials in two variables (x and y for faces parallel to the xy plane). These polynomials contain terms of the form $x^a y^b$ where $0 \leq a, b \leq 3$. This gives us 16 polynomial coefficients. This leads to a 4×4 grid of sample points on each face, where values on edges and vertices are shared across neighboring faces. The canonical cube is shown in Figure 3.1. This leads to 56 sample points for the full cube with 168 total degrees of freedom. We must therefore construct a basis with 168 functions, of which 167 will be divergence-free and one will be divergent.

The basis construction for the cube case mirrors the tetrahedron case. We begin by identifying a maximal set of polynomial basis functions. We are able to construct 135 divergence-free modes and the divergent mode as polynomials. This leaves 32 modes that must be divergent. In 2D, these denominators looked like $x + y$ (up to symmetries such as $x \rightarrow 1 - x$ and $y \rightarrow 1 - y$). In 3D they look like $(x + y)(x + z)(y + z)$ up to symmetries. While 32 may seem like a lot of rational modes to compute, most of these are related by simple symmetries. There are only three novel basis modes required for the cube; the rest are obtained from these by symmetries. This makes them much simpler to implement. The procedure for finding them is similar to the procedure for tetrahedra. Our final cube basis is shown in Tables 3.7 and 3.8.

3.2.2 Basis selection

The process for constructing an interpolation function from sample points and a set of basis modes closely follows the procedure in 2D. Each basis function $\bar{\mathbf{u}}_k$ is associated with a sample vector $\hat{\mathbf{u}}_k$, which consists of the values of the basis function evaluated at the M sample points and stacked into a large vector of size $N = 3M$. The $N \times N$ matrix $\mathbf{K}$ is constructed by taking the vectors $\hat{\mathbf{u}}_k$ as its columns. By construction, the basis vectors $\hat{\mathbf{u}}_k$ are linearly independent, so that $\mathbf{K}$ is invertible. Since it only depends on the basis vectors, it is also fixed. The interpolation function is a linear combination of the basis functions

$$\bar{\mathbf{u}}(x, y, z) = \sum_{k=1}^N c_k \bar{\mathbf{u}}_k(x, y, z). \quad (3.2)$$

This function should agree with the prescribed sample points $\hat{\mathbf{w}}$, so that

$$\hat{\mathbf{w}} = \sum_{k=1}^N c_k \hat{\mathbf{u}}_k = \mathbf{K}\mathbf{c}, \quad (3.3)$$

The coefficients c_k are then $\mathbf{c} = \mathbf{K}^{-1}\hat{\mathbf{w}}$. Since $\mathbf{K}$ is fixed, we can simply precompute and hard-code the matrix $\mathbf{K}^{-1}$. We thus obtain our coefficients from our sample points with a fixed-size matrix-vector multiply.

3.3 Cut-cell div-free interpolation

With our divergence free interpolation on cubes and tetrahedra, we can now extend an existing spline-based divergence-free interpolation scheme on MAC grids to irregular boundaries. The general strategy here is the same as in 2D, though the 3D case is more

difficult in several places. As in 2D, the starting point is a MAC grid, where normal velocity components are stored at grid faces.

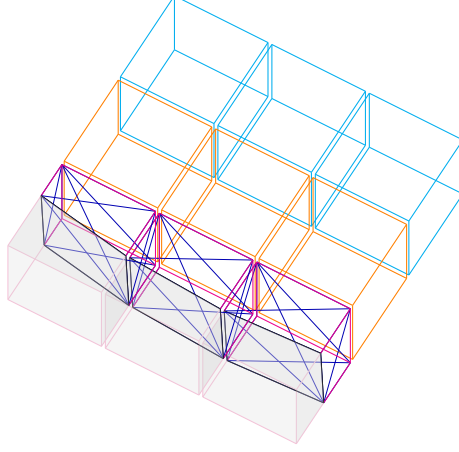


Figure 3.2: Cell classification in a $3 \times 3 \times 1$ portion of the domain. () indicates the interior cells, which are uncut cells with uncut neighbors; spline-based interpolation is possible on these cells. () indicates the transition cells, which are uncut cells with cut neighbors; these are interpolated as cubes. The cut-cells () are grid cells that are cut by the boundary; these are divided into tetrahedra. The portions of the cut cells outside the domain are shaded gray.

This grid data has been projected to be discretely divergence-free. We assume that Ng et al. [63] was used for this purpose, though any scheme that defines its discrete notion of divergence free in the same way is suitable. Ng et al. [63] defines for each face a *face fraction*, which is the fraction of the area of the face (between 0 and 1) that is inside the cut boundary. It is 0 for uncut faces that are completely outside the domain and 1 for uncut faces that are inside. For faces that have been cut, we compute this face fraction using marching cubes. We make use of a carefully-crafted marching cubes table for this purpose, which we describe in Section 3.4. The velocity components stored on the MAC grid represent the *integrated average* velocity over the portion of the grid face that is inside. Multiplying this by the face fraction gives us the net flux of fluid across that face. Ng et al.

[63] projects the MAC grid data so that the net flux of fluid into each grid cell is zero. Note that in the case of nonzero boundary conditions, the flux across the boundary also contributes to the net flux for the cell.

For grid cells that are far enough from the boundary, one may use an efficient spline-based divergence-free interpolation scheme. As in 2D we require the spline-based scheme to additionally be *flux consistent* in the sense that the integrated average normal velocity component over a grid face should agree with the MAC sample stored on that face. We use the flux consistent scheme from Roy-Chowdhury et al. [67]. This scheme uses MAC data from a $3 \times 3 \times 3$ cell neighborhood in order to interpolate within a cell; this scheme can only be used if none of those cells is cut. For all other cells, we must interpolate using cubes or tetrahedra. To facilitate this, we classify cells based on how they will be interpolated.

3.3.1 Cell classification

The cell classification is the same as in 2D. Cells that are fully *outside* of the domain are not interpolated. Cells that have been cut by the boundary are called *cut* cells; these cells will be cut into tetrahedra using marching cubes and then interpolated as tetrahedra. *Transition* cells are not cut cells but are adjacent to at least one cut cell; these cells will be interpolated using cubes. *Interior* cells lie entirely within the domain and are not adjacent to any cut cells; these cells are interpolated using the efficient spline-based scheme. The classification is shown in the Figure 3.2.

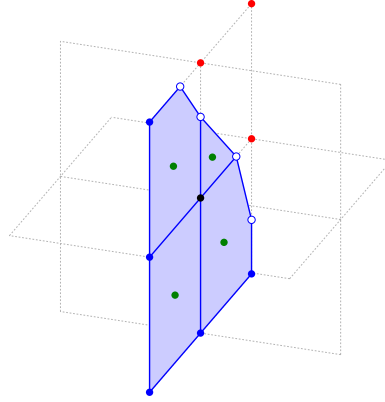


Figure 3.3: Velocities at transition nodes are reconstructed from neighboring face data. For each coordinate direction, the nodal normal velocity is obtained by a least-squares fit using four neighboring faces, whose MAC grid components are taken to live at the centroid of the interior portion of the grid face. For the emphasized direction, the interior portions of the face edges (—), the interior node (●), and exterior vertices (●) are shown. The four face centroids used in the fit are marked in green (●), and the node being computed is marked in black (●). The other two coordinate directions (.....) are omitted for clarity; in total, twelve faces are used for each node (four per velocity component).

3.3.2 Node classification

Next, we classify nodes. Our node classification is the same as in 2D, but we must be more careful when computing them. *Interior nodes* are corners of at least one interior cell. The velocity at the interior nodes is computed using the spline-based scheme to ensure continuity. *Outside nodes* lie outside the interface and are never used. All other grid nodes are *transition nodes*. These must be computed from the MAC velocity data from the surrounding 12 faces, all of which contain valid data.

In 2D, the faces are edges, so their geometry is entirely determined by their face fractions. This allowed transition nodes to be evaluated by weighting the surrounding MAC grid data using the face fractions. Applying a similar strategy in 3D results in an interpolation scheme that fails to recover affine velocity fields. In 3D, the node velocity requires three components, each of which is stored on a different set of MAC grid faces. Each

component is interpolated from the four adjacent MAC grid faces, as shown in Figure 3.3. The MAC grid data stores an integrated average of the velocity on the interior portion of a grid face. If the exact velocity is affine, then this averaged velocity will agree with the analytical velocity at the centroid of the interior portion of a grid face. This gives us an accurate velocity component at the centroid locations for the four grid faces. We compute the nodal velocity component from the four centroid components using linear regression. This requires solving a small 3×3 linear system for each of the components on the transition nodes.

In addition to grid nodes, we also add *cut points*, which are the locations where the grid edges intersect the domain boundary. The velocities at the cut points are computed from the domain boundary conditions. The node classification is shown in Figure 3.4.

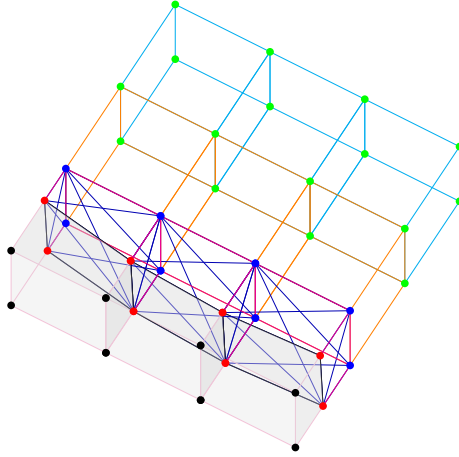


Figure 3.4: Node classification within a $3 \times 3 \times 1$ region of the computational domain. (●) are interior nodes; these are nodes that touch interior cells. The nodes outside the domain are outside nodes (●). The cut points (●) are located on edges where they intersect the boundary. The rest are the transition nodes (●).

3.3.3 Edge classification

We next compute sample points along the edges of our canonical shapes. For transition cells, we will require two sample points along segments. For tetrahedra, we will require three sample points along edges. *Interior edges* are edges of the MAC grid that are also edges of an interior cell; sample points on these edges are computed by evaluating the spline-based scheme to ensure continuity. *Transition edges* are uncut edges of the MAC grid that are not adjacent to an interior cell; sample points on these edges are computed by linearly interpolating the velocities stored at the endpoints. All of the edges that occur on transition cells are interior edges or transition edges. *Diagonal edges* are diagonal edges on grid faces between transition cells and cut cells; these sample points are used to define the interpolation inside the adjacent cut cell. These sample points can be computed by interpolating the velocity from the transition cell. In practice, we find it more convenient to compute them directly, as noted in Section 3.3.6. *Cut edges* are the remaining edges. These include all of the other edges of the tetrahedra inside cut cells, such as cut grid edges or edges in the interiors of cut cells. Sample points on these edges are computed by linearly interpolating the velocities stored at the endpoints. The edge classification is shown in Figure 3.5.

3.3.4 Face classification - cubes

Our cube discretization uses tensor product cubic polynomials along each face. This requires 16 degrees of freedom for each face of the cube, of which 4 are supplied by the vertices and 8 are supplied along the four edges. This leaves 4 sample points that must

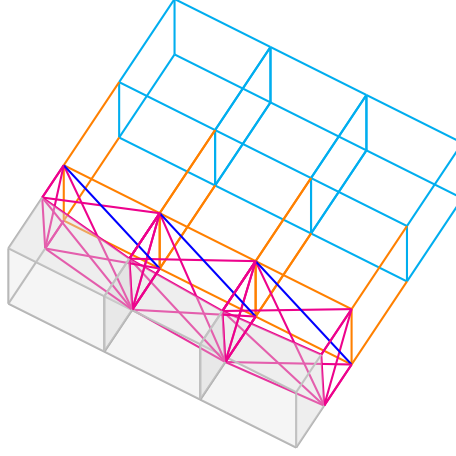


Figure 3.5: Classification of edges in a $3 \times 3 \times 1$ portion of the domain. (—) indicates the interior edges, which are edges that touch an interior cell. (—) indicates the transition edges, which border a transition cell but not an interior cell. The diagonal edges (—) are diagonals of the square faces between the cut-cells and transition cells. Other edges inside the domain are cut edges (—).

be computed in the interior of each cube face. At this point, we have already computed sample points at the vertices and edges of these faces. *Interior faces* are grid cell faces that are adjacent to an interior cell; the sample points on these faces are computed using the spline-based interpolation scheme for interior cells. The rest of the grid faces that border transition cells we refer to as *transition faces*. We compute sample points for these faces from the MAC grid sample for the face as well as the sample points on the edges and vertices of the face.

Transition faces come in two slightly different varieties. In general, one or more of the edges of a transition face may touch an interior face, and those sample points are computed from the spline-based interpolation scheme. Faces of this type always occur between two transition cells. *Simple* transition faces do not contain any edges that touch interior cells; these edges are always linearly interpolated from their endpoints. All transition faces

that touch cut cells are simple, and the way that their sample points are constructed plays a major role in the treatment of tetrahedra.

In the absence of flux considerations, we could define the velocity field inside a simple transition face using bilinear interpolation from the vertices. This definition is consistent with doing linear interpolation along the edges of the face. For z faces, this gives us the polynomial basis $1, x, y$, and xy along the face. We use this to define the velocity field in the tangential direction, but it does not give us enough freedom in the normal direction. The flux across the face would be entirely determined from linear interpolation and corresponds to the average of the vertices. The actual flux across the face is stored in the MAC grid data and will be different from this value. Instead, we must supplement our polynomial basis $1, x, y, xy$, and $x(1-x)y(1-y)$. The additional mode gives us the ability to specify an arbitrary flux along the face while being consistent with the linear interpolation along the edges. Velocity fields on simple transition faces are always polynomials of this form. Note that the maximum polynomial degree on a simple transition face is 4; this is why we needed to define our canonical tetrahedron basis to support degree-4 polynomials.

With the special needs of simple transition faces in mind, we return to the general discussion of transition faces. We have available to us 13 velocity values: 4 vertex sample points, 8 edge sample points, and the integrated average velocity. In the normal direction, the integrated average is the MAC grid velocity component. In the tangential direction, the integrated average is a design choice, and we use the average of the other 12 values. This gives us the correct integrated average in the special case of simple faces and is observed to work well in the general case. This gives us our 13 input velocities. The output is 4 sample points for the face. We would like the mapping from input to output to be linear, so our

task is to construct a 4×13 matrix. Once we have this matrix, computing sample points on transition faces becomes a matrix-matrix multiplication (with sizes 4×13 and 13×3 , since sample points are 3D vectors).

We begin the construction of the 4×13 matrix by first finding a 16×13 matrix, which maps the 13 input values to the 16 coefficients of a general bi-cubic polynomial $p(x, y)$ on the square domain $[0, 1]^2$. We then enforce properties that this polynomial must satisfy:

- $p(x, y)$ agrees with the first 12 input values at the locations of the sample points.
- The integrated average $\int_0^1 \int_0^1 p(x, y) dx dy$ agrees with the last input value.
- If the edge sample points are linearly interpolated from the vertex sample points, then $p(x, y) = c_0 + c_1x + c_2y + c_3xy + c_4x(1-x)y(1-y)$ for some coefficients c_k . This ensures that simple faces are handled correctly.
- If the sample points are consistent with a polynomial $p(x, y)$ that is quadratic in x (or y), then the reconstructed $p(x, y)$ will be quadratic in x (or y).

Note that the final requirement is not strictly needed by our algorithm, but it serves as a convenient way to choose some of the remaining degrees of freedom (in fact, the fourth property implies the third). This still leaves 4 degrees of freedom, which we choose arbi-

trarily. We can then evaluate this polynomial at the 4 sample points. This gives us the 4×13 matrix

$$\mathbf{M}_{tf} = \frac{1}{9} \begin{pmatrix} -2 & 0 & 0 & 1 & 3 & -3 & 0 & -3 & 3 & -3 & 0 & -3 & 16 \\ 0 & -2 & 1 & 0 & -3 & 3 & -3 & 0 & 0 & -3 & 3 & -3 & 16 \\ 0 & 1 & -2 & 0 & 0 & -3 & 3 & -3 & -3 & 3 & -3 & 0 & 16 \\ 1 & 0 & 0 & -2 & -3 & 0 & -3 & 3 & -3 & 0 & -3 & 3 & 16 \end{pmatrix}. \quad (3.4)$$

The indexing of the 13 input degrees of freedom and the 4 output degrees of freedom are shown in (a) of Figure 3.6. Note that we used the same procedure for simple and non-simple transition faces.

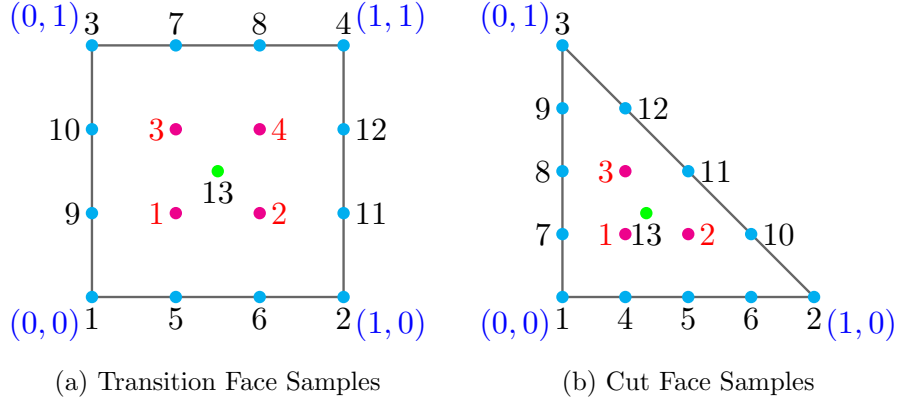


Figure 3.6: (a) Ordering of samples on a transition face corresponding to (3.4). The black number indicates the index of corresponding matrix column for each input sample position; the green entry represents the integrated average across the square. The red number indicates the index of corresponding matrix row for each output face sample. (b) Ordering of samples on a triangle face corresponding to (3.5). The black number indicates the index of corresponding matrix column for each input sample position. The green entry is the integrated average across the triangle. The red number indicates the index of corresponding matrix row for each output face sample.

At this point in the algorithm, we have all of the information we need to perform divergence-free interpolation in transition cells using our canonical cube. All that remains is to handle cut cells.

3.3.5 Face classification - tetrahedra

The treatment of cut cells is complicated by the need to manage the flow through the tetrahedra within a cut cell. The first step is to compute the flux across each of the triangles that forms a face of a tetrahedron. This process couples the tetrahedra and is somewhat sensitive to the marching cubes discretization. Once these fluxes are available, we use the fluxes to compute sample points on the faces; this step is independent for each triangle.

Face fluxes

We use marching cubes to divide each cut cell into tetrahedra as described in Section 3.4. We evaluate flux across the boundary from domain boundary conditions. Additionally, we have the total flux across each of the faces of the cube. This allows us to compute the total flux into the cell. We divide this total net flux among the tetrahedra in proportion to their volumes; this prevents the scheme from concentrating all of the divergence into one tetrahedron. If the flow is incompressible, the total flux should be zero.

The next step is to compute fluxes for the triangles that lie along the faces of the grid cell. For triangles that border transition cells, the velocity field (and thus the flux) along the triangle is given by the interpolation scheme in the transition cell.

For triangles that lie on the boundary between two cut cells, we have some freedom. Grid faces between cut cells are cut into triangles using marching cubes. An obvious solution would be to divide the total flux across the grid face to the triangles in proportion to their area, but this solution would not recover affine fields correctly. Instead, we use a two-step procedure. First, we estimate the integrated average face velocity for each of the triangles as the average of the velocities at its vertices; scaling by area gives a flux estimate. In the case of an affine velocity field, the vertices at the velocities will be exact, so the integrated average will also be exact. Adding the estimated fluxes for the triangles gives us the total estimated flux, which will in general differ from the actual flux across the grid face. We then divide the residual flux among the triangles in proportion to their areas. This gives us our fluxes for these triangles. Note that in the case of an affine velocity field, the estimated flux will equal the actual flux across the face, and so the correction step has no effect.

What remains is to compute fluxes for triangles that are in the interior of the grid cell. Following the strategy similar for 2D, we use a greedy strategy. For tetrahedra where three of the four triangles have fluxes available, we compute the flux for the fourth triangle so that the target flux in the tetrahedron is achieved. In 2D, all of the fluxes can be computed in this way, but in 3D this is not the case. One will often encounter a situation where all remaining tetrahedra have at most two triangles with valid fluxes. We can break the deadlock by selecting a tetrahedron with fluxes on only two triangles and assign one of the other triangles a *symbolic* flux. The tetrahedron now has three faces with fluxes, and the algorithm continues. The number of symbolic fluxes that may be required depends on the marching cubes table, but in our case the maximum number of symbolic fluxes that may be required is three. That is, instead of fluxes being a simple number, the fluxes have

the form $a + bx + cy + dz$, where a, b, c, d are numbers and x, y, z are free variables that must be determined. At this stage, all of the triangles have fluxes, but we may have 0-3 variables that must be selected. For each of the interior triangles, we compute an estimated flux by averaging the vertex velocities and multiplying by triangle area. We then compute x, y, z to minimize the difference between the resulting triangle fluxes and the estimated fluxes in the least squares sense. This requires us to solve one linear system of size up to 3 for each cut cell. With x, y, z available, we can compute the actual numerical flux for each triangle. Dividing the triangle fluxes by triangle area gives us the integrated average velocity for each triangle. With this flux computation, the per-triangle fluxes enable the interpolation to exactly recover the affine velocity field.

Sample points

At this stage, each triangle inside a cut cell has sample point available at each vertices, three sample points along each edge, and the integrated average velocity for the triangle. We need to compute three sample points for the interior of the triangle. In the case of triangles that are on grid faces touching transition cells, the sample points can simply be evaluated from the transition cell. For the other triangles, we compute the sample points using a matrix-matrix multiply in much the same way we did in Section 3.3.4 for transition cells. Each triangle has a sample point on each vertex, three sample points on each edge, and an integrated average velocity. This gives us 13 velocity values for each triangle; see (b) in Figure 3.6 for the numbering of the 13 inputs and the three outputs. The three outputs

are the three sample points for the triangle. The output is produced by multiplying by the 3×13 matrix

$$\mathbf{M}_{tf} = \frac{1}{9} \begin{pmatrix} 0 & 0 & 0 & -4 & 9 & -12 & -4 & 9 & -12 & 4 & -15 & 4 & 45 \\ 0 & 0 & 0 & -12 & 9 & -4 & 4 & -15 & 4 & -4 & 9 & -12 & 45 \\ 0 & 0 & 0 & 4 & -15 & 4 & -12 & 9 & -4 & -12 & 9 & -4 & 45 \end{pmatrix}. \quad (3.5)$$

This matrix is computed in a similar way as in Section 3.3.4. We construct from the 13 inputs the coefficients of a general degree-4 polynomial for the triangle. We require that evaluating the polynomial at the sample points recovers the first 12 inputs, and computing the integrated average over the triangle recovers the last input. To help eliminate degrees of freedom, we then require that if the triangle has edges defined by linear interpolation, then the resulting polynomial will have the form $c_0 + c_1x + c_2y + c_3xy(1 - x - y)$. This case is very common; all triangles that do not share edges with the boundary or transition cells are of this type. In the case of an affine velocity field, this ensures that the sample points will be computed exactly. The remaining degrees of freedom are chosen arbitrarily.

3.3.6 Direct computation from transition faces

Edge sample points, triangle integrated average velocity, triangle fluxes, and triangle sample points that occur on grid faces touching transition cells are all defined by interpolation in the transition cell. This definition is necessary for continuity. While this is a simple and perfectly reasonable way to compute these quantities, we in practice opt to compute them directly since we construct tetrahedra in cut cells and cubes in transition cells during the same pass through our grid. The computational procedures below are

equivalent to interpolation in the transition cell. We begin with a grid face that is adjacent to a transition cell and a cut cell. This face is the square $ABCD$, as shown in Figure 3.7. Marching cubes divides this square into two triangles; we will consider the triangle ABC , where BC is a diagonal of the square. Let u_A , u_B , u_C , and u_D be the velocities at the four vertices of the grid face. Let u_E be the integrated average velocity for the face. The normal component of u_E is just the sample stored on the MAC grid face at that location; the tangential components are the average of the four vertices. All of the desired quantities are linear combinations of these five values with fixed coefficients, which makes them very efficient to compute. The coefficients are deduced so that these procedures are equivalent to interpolation in the transition cell.

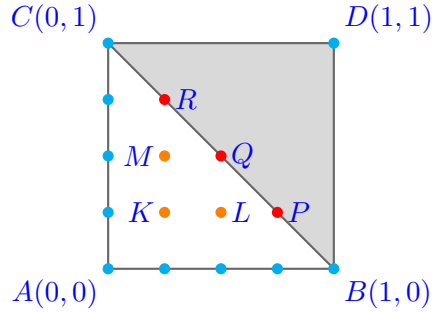


Figure 3.7: The square face between a transition cell and a cut cell is split along its diagonal into two triangles. This illustration shows the indexing convention used in Section 3.3.6 and shows the spatial location of sample points. The polynomial on the triangle is defined by interpolation from the neighboring transition cell that shares this face. Samples on the face and along the diagonal edge are computed from the velocities at the four square vertices and the average velocity integrated over the square face. Other edge samples are linearly interpolated.

Let u_P , u_Q , and u_R be the three sample points on the BC , the diagonal of the square $ABCD$, as shown in Figure 3.7. Then,

$$u_P = \frac{-33u_A + 63u_B - 65u_C - 33u_D + 324u_E}{256} \quad (3.6)$$

$$u_Q = \frac{-80u_A - 80u_B - 80u_C - 80u_D + 576u_E}{256} \quad (3.7)$$

$$u_R = \frac{-33u_A - 65u_B + 63u_C - 33u_D + 324u_E}{256}. \quad (3.8)$$

The integrated average velocity for triangle ABC is

$$u = u_F + \frac{u_A - u_D}{6}. \quad (3.9)$$

The flux across this triangle is obtained by multiplying the integrated average by the triangle's area. Finally, the sample points u_K , u_L , and u_M inside the triangle (See Figure 3.7) are

$$u_K = \frac{63u_A - 33u_B - 33u_C - 65u_D + 324u_E}{256} \quad (3.10)$$

$$u_L = \frac{-12u_A - 12u_B - 76u_C - 76u_D + 432u_E}{256} \quad (3.11)$$

$$u_M = \frac{-12u_A - 76u_B - 12u_C - 76u_D + 432u_E}{256}. \quad (3.12)$$

3.3.7 Interpolation procedure

All of the sample points for divergence-free cubes and divergence-free tetrahedra are now available. The basis function coefficients are computed from the sample points via matrix multiplication as described in Section 3.2.2. This completes the interpolation setup.

To interpolate to a specified location in space, we first identify the grid cell that contains that point and determine whether it is an interior cell, transition cell, cut cell, or outside cell. Outside cells are not interpolated. Interior cells are interpolated from the spline-based scheme. Transition cells are interpolated using a divergence-free cube. For cut cells, we visit each tetrahedron in the cell to find the one that contains the query point and then use a divergence-free tetrahedron for interpolation.

3.4 Marching cubes with tetrahedra

In this paper, we use marching cubes for in our discretization of Ng et al. [63], where it is used to calculate face areas. We also use it to construct a tetrahedralization for the interior of cut grid cells. Each grid cell has 8 vertices, and the level set at each vertex may be positive or negative, resulting in $2^8 = 256$ cases. A traditional marching cubes table encodes for each of these cases the topology of the triangles that form the *surface* between the inside and outside portions of a grid cell. One may construct for each of the six faces of the grid cell a triangulation of the interior portion of that face that is inside the level set. We will refer to this as the *boundary* triangulation. The surface and boundary triangulations together form a polyhedron, which can be divided into tetrahedra. We will refer to this as the *interior* tetrahedralization. All three discretizations can be encoded using a marching cubes style tabulation. These discretizations are shown in Figure 3.8. To be suitable, a marching cubes implementation must satisfy the following properties:

- The surface triangles must be faces of the interior tetrahedra.
- The boundary triangles must be faces of the interior tetrahedra.

- The interior tetrahedra must always have nonnegative volume for any values of the level set.
- The boundary triangles must be consistent across neighboring grid cells.

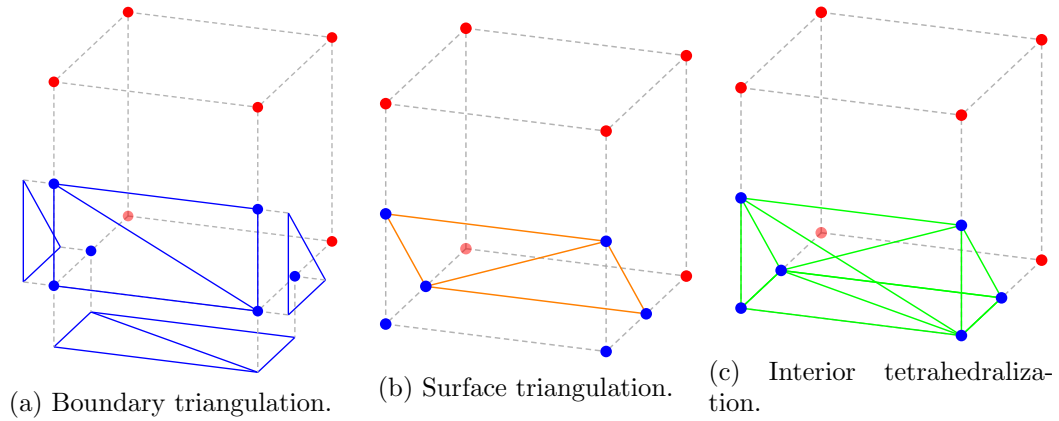


Figure 3.8: For each marching cubes case, the marching cubes table encodes the boundary triangulation (left: $\triangle$), surface triangulation (middle: $\triangle$), and interior tetrahedra (right: $\triangle$). Boundary triangles and surface triangles are faces of the interior tetrahedra. Boundary triangles lie on faces of the cube. Boundary triangles on a cube face are identical for both cells that share the cube face.

We also do not allow additional vertices to be inserted other than the cut points along the edges of the cube and the vertices of the cube. We are not aware of any previously existing marching cubes implementation with these properties. Indeed, these properties are nontrivial to achieve at the same time. The nonnegative volume requirement forbids certain tetrahedra, such as the one in Figure 3.9 from occurring in the table. Because many of the boundary triangles are coplanar, many of the possible polyhedra cannot be tetrahedralized at all without inserting new vertices; see Figure 3.10 for an example. If the cases are tetrahedralized independently, they will not match up with the tetrahedra in neighboring

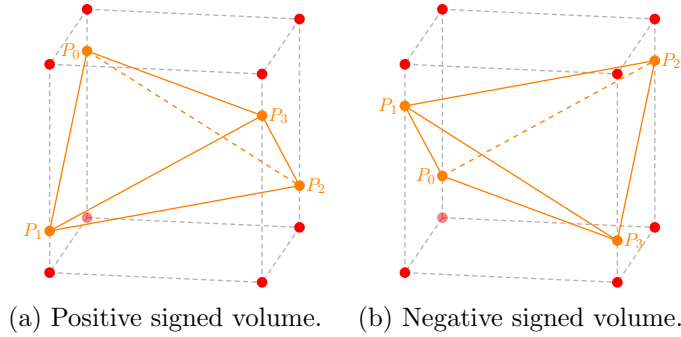


Figure 3.9: This is an example of a forbidden tetrahedron under the non-negative volume requirement. The sign of the volume of this tetrahedron can change depending on where edges are cut. Such tetrahedra are not permitted in the marching cubes table.

cells. Unlike in classical marching cubes, the solutions to the cases are not independent.

All cases must be constructed at once so that they agree at the boundaries.

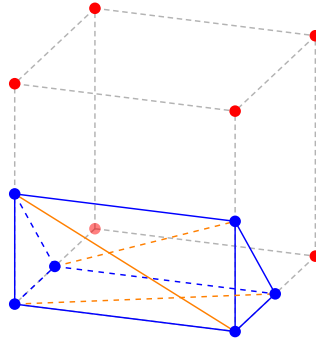


Figure 3.10: Example of a polyhedron with its surface triangulation and boundary triangulation shown. Several boundary triangles are coplanar, which prevents the polyhedron from being tetrahedralized into valid nonnegative-volume tetrahedra using only the existing vertices. Additional vertices must be inserted to obtain a valid tetrahedralization. Polyhedra such as this one cannot be represented by a tetrahedralization in the marching-cube lookup table. The marching cubes table must be constructed to avoid cases such as this.

3.4.1 Cube face triangulation

The key to achieving continuity is to fix the triangulation along the faces of the cubes. Each cube face has 4 vertices, resulting in $2^4 = 16$ different cases. Each case must be triangulated. Of these, one case produces no triangles (case 0), four produce one

triangle (cases 1, 2, 4, 8), and two produce isolated triangles (cases 6 and 9). For these cases, the triangulation is fixed. Five cases produce a quadrilateral (cases 3, 5, 10, 12, 15), which can be divided into triangles in 2 ways. The remaining four cases produce a pentagon (cases 7, 11, 13, 14), which can be divided into triangles in 5 ways. This leads to $2^5 \cdot 5^4 = 20,000$ possible ways in which the boundary triangulations can be chosen. We select a suitable triangulation for the boundary by brute force. For each of the 20,000 possible ways of selecting the boundary triangulation, we attempt to compute tetrahedra for all 256 marching cubes cases.

3.4.2 Tetrahedralization procedure - marching cubes

We begin with an existing marching cubes implementation, which gives us interface triangles for all 256 cases. By fixing one of the 20,000 boundary triangulations, we are left with a polyhedron for each of the 256 cases. We must then divide the polyhedron into tetrahedra. In the process of doing this, we must ensure that we do not violate any of the criteria above. We also seek to minimize the number of symbolic variables required during the flux computation stage of our interpolation scheme.

The vertices of all of the tetrahedra must lie on a vertex of the cube or at a cut point along an edge of the cube. Since a cube has 8 vertices and 12 edges, there are 20 possible tetrahedron vertex locations and thus $\binom{20}{4} = 4845$ possible tetrahedra. We then discard tetrahedra that always have zero volume or for which the volume's sign depends on where along the edges the cube was cut. This gives us a list of valid tetrahedra.

We construct the tetrahedra by a recursive backtracking algorithm. At each level, we identify all pairs of adjacent triangles on the polyhedron whose vertices form a valid

tetrahedron. For each such tetrahedron, we try removing the tetrahedron from the polyhedron and attempt to divide the result into tetrahedra recursively.

When three of the faces of the chosen tetrahedron are on the surface of the polyhedron, the tetrahedron will be solvable. The order in which the backtracking constructs tetrahedra is also a suitable order for solving the tetrahedra to compute triangle fluxes; fluxes for all triangles on the surface of the polyhedron are known. Thus, the process of removing the polyhedron is equivalent to the process of computing the flux on the fourth face. Removing these tetrahedra results in removing three triangles and adding back one (unless the polyhedron itself is a tetrahedron, in which case we are done). Thus, we are guaranteed progress.

When only two of the faces of tetrahedron are on the surface of the polyhedron, the tetrahedron is not solvable. While computing fluxes, we had to add a symbolic flux to solve such tetrahedra. During the tetrahedralization procedure, we enforce a maximum number of times we are allowed to select this type of tetrahedron; after that they are discarded. Note that removing these tetrahedra leaves the number of triangles in the polyhedron unchanged. It is also possible that “removing” the tetrahedron would actually make the polyhedron larger. Setting a maximum number of such tetrahedra guarantees progress, and final correctness is ultimately ensured by manual inspection of the final table.

Without additional refinements, the above algorithm results in combinatorial explosion. We can prevent this by keeping track of the tetrahedra that have been selected or rejected; these tetrahedra are not eligible to be selected during recursive calls. This prevents tetrahedra from being added and removed repeatedly (no termination), and it prevents us

from trying to add the same set of tetrahedra in all possible orderings (combinatorial explosion).

To minimize the number of symbolic fluxes, we first attempt to construct tetrahedra with a maximum limit of zero fluxes. If that fails, we try again allowing one flux, then two, and then three fluxes. We found a solution with only three fluxes required, so we need not search beyond that.

The procedure we used for constructing tetrahedra is not an efficient algorithm, nor is it guaranteed to find a solution if one exists. It is not a general purpose algorithm for constructing tetrahedra. It did suffice for our purposes. Of the 20,000 possible boundary triangulations for the cases, *two* yielded valid tetrahedra for all 256 cases. Although the algorithm does not technically guarantee that the tetrahedralization is suitable, we verified all 256 cases manually to ensure that it is. The boundary triangulations we used is shown in Figure 3.11.

Since the need for continuity between faces prevents the table from being filled using symmetries, all cases are unique. Instead, please see the source code for this project, which is available online from the corresponding author's website, for the full marching cubes implementation, along with the marching cubes tables and corresponding tables of tetrahedra. This includes the code used to extract triangles and tetrahedra from the tables.

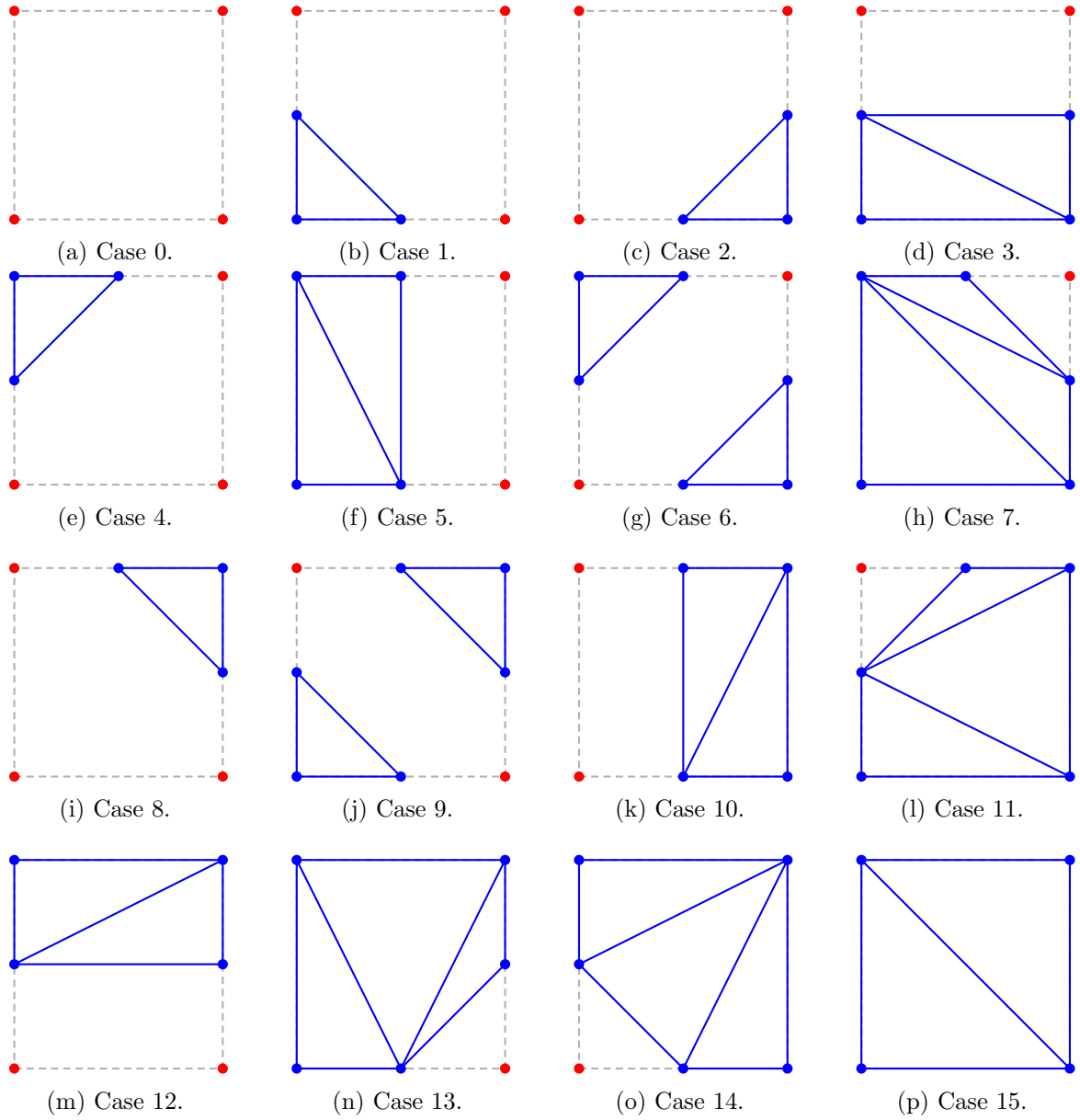


Figure 3.11: Boundary triangulations for all 16 cases. For faces parallel to the xy plane or the xz plane, the x axis runs horizontally and the other axis runs vertically. For faces parallel to the yz plane, the y axis runs horizontally and the z axis vertically. The node with the lowest index is the bottom left vertex. The same case layout is used for all MAC grid faces, which causes the discretizations in neighboring cubes to be consistent.

3.5 Numerical tests

In the numerical tests that follow, we use the following vector fields

$$\begin{aligned}
\mathbf{u}_{3a} &= \begin{pmatrix} \sin(x+2) \sin(y+4) \sin(z+6) \\ \cos(x+2) \cos(y+4) \cos(z+6) \\ \cos(x+2) \sin(y+4) (\cos(z+6) + \sin(z+6)) \end{pmatrix} \\
\mathbf{u}_{3b} &= \begin{pmatrix} \sin(370x+2) \sin(370y+4) \sin(370z+6) \\ \cos(370x+2) \cos(370y+4) \cos(370z+6) \\ \cos(370x+2) \sin(370y+4) (\cos(370z+6) + \sin(370z+6)) \end{pmatrix} \\
\mathbf{u}_{3c} &= \begin{pmatrix} \sin(x+2) + \sin(y+4) \\ \cos(y+2) + \cos(z+4) \\ \cos(z+2) + \cos(x+4) \end{pmatrix} \\
\mathbf{u}_{3d} &= \begin{pmatrix} x+2y \\ y+z \\ -2z+3x \end{pmatrix} \\
\mathbf{u}_{3e} &= \begin{pmatrix} x+2y+3z+1 \\ 4x+2y+5z-2 \\ 6x+7y-z+3 \end{pmatrix}.
\end{aligned}$$

The fields $\mathbf{u}_{3a}$, $\mathbf{u}_{3b}$ and $\mathbf{u}_{3c}$ are from [67]. The field $\mathbf{u}_{3a}$ is divergence-free, well-resolved, and nonlinear; it provides a good overall evaluation of accuracy. The field $\mathbf{u}_{3b}$ is divergence-free, very high frequency, and nonlinear; it is not well-resolved at any of the resolutions we test and behaves essentially as random numbers. The field $\mathbf{u}_{3c}$ is nonlinear and well-resolved

but not divergence-free; it provides a good overall evaluation of accuracy on divergent fields. The final two fields $\mathbf{u}_{3d}$ and $\mathbf{u}_{3e}$ are affine, with $\mathbf{u}_{3d}$ being divergence-free and $\mathbf{u}_{3e}$ being divergent. We use them to evaluate our ability to exactly recover divergence-free fields. Most of our numerical tests are based on the tests used for the 2D version of this algorithm [44].

3.5.1 Random tetrahedron and random cube test

Next we demonstrate that interpolation over a individual tetrahedron or cube is continuous at the boundary. We also verify that they produce divergence-free interpolation when the boundary conditions have zero net flux. We perform this test using a randomly generated (C)ube or (T)etrahedron and a randomly generated velocity field that is divergence-(F)ree or (D)ivergent. This leads to four case. We sample tetrahedron vertices in $[-1, 1]^3$, and for cubes we select the side length randomly in $[0.1, 2]$. We randomly generate element boundary conditions by choosing sample points in $\|\mathbf{u}\|_\infty \leq 1$. In the divergence-free case, we project off the divergent mode.

To verify continuity, we need to show that the interpolated velocity agrees with the prescribed values at the boundary. To do this, we sample 100 points on each face of the tetrahedron or cube. At each point, we compute the velocity directly from the polynomial on that face of the element and compare it to the velocity obtained by interpolation. The ℓ_∞ norm of the error is shown as $\ell_\infty(\text{err})$ in Table 3.1. In all four cases the errors are at

roundoff levels. We test the divergence-free property by sampling 100 points in the interior.

At each point, we estimate the divergence with finite differences using

$$\begin{aligned} \nabla \cdot \mathbf{u} \approx & \frac{u(x + \epsilon, y, z) - u(x - \epsilon, y, z)}{2\epsilon} + \frac{v(x, y + \epsilon, z) - v(x, y - \epsilon, z)}{2\epsilon} \\ & + \frac{w(x, y, z + \epsilon) - w(x, y, z - \epsilon)}{2\epsilon} \end{aligned} \quad (3.13)$$

with $\epsilon = 10^{-6}$. The results are shown as $\ell_\infty(\text{div})$ in Table 3.1. When the boundary conditions are projected to be divergence free, the maximum divergence on the order of 10^{-7} . This error is due to the accuracy of the finite difference approximation. These errors are not at roundoff error due to truncation in the finite difference approximation, which is especially high near the corners of the shapes where the interpolation is continuous but not differentiable. This is similar to what is observed in the 2D case [44]. When the boundary conditions are not projected to be divergence-free, the maximum divergence is finite and on the order of unity. Note that continuity is achieved even if the velocity field is not divergence-free.

Shape type	Case label	$\ell_\infty(\text{err})$	$\ell_\infty(\text{div})$
Tetrahedron	TF	2.43×10^{-13}	1.76×10^{-7}
	TD	2.52×10^{-13}	0.983
Cube	CF	4.02×10^{-12}	8.99×10^{-7}
	CD	6.05×10^{-12}	0.176

Table 3.1: Results of the numerical continuity tests. Here, “T” denotes tetrahedra and “C” denotes cubes. The labels “F” and “D” indicate whether the velocity field is divergence-free or divergent, respectively. In all cases, the continuity error on the boundary faces is at the level of roundoff. For the divergence-free cases, the measured divergence error is at the truncation-error level of the central-difference stencil used to compute it.

3.5.2 Seeding MAC grid data

In subsequent numerical tests, we work from MAC grid data. Our interpolation scheme assumes that MAC grids will store the normal component of the integrated average velocity for each face of the grid. This is the state that the MAC grid would naturally be in after projected using a scheme such as [63]. When initializing the grid from an analytical velocity, it is important that the MAC grid be initialized to this state. For MAC faces, we use marching cubes to get a triangulation of the interior portion of the face. We then compute the integrated average velocity over the triangles analytically. Scaling these averages by the triangle areas gives us fluxes, which are additive. Dividing the total face flux by the total face area produces the final integrated average velocity that we store on MAC grid faces. For domain boundary conditions, our algorithm uses integrated average velocities over a triangulation of the portion of the boundary within the grid cell. We construct this triangulation using marching cubes and compute the integrals the same way as for the MAC grid faces. We use the same procedure for the boundary conditions required by the projection of [63].

This leaves us with the problem of integrating analytical velocity fields exactly over triangles. The velocity fields $\mathbf{u}_{3d}$ and $\mathbf{u}_{3e}$ are polynomials, so we can simply use Gaussian quadrature to compute them. The remaining velocity fields contain terms like $\sin(ax + r) \sin(by + s) \sin(cz + t)$, $\cos(ax + r) \sin(by + s) \cos(cz + t)$, etc. We use angle sum and difference formulas to compute these quantities in terms of quantities of the form

$\sin(ax + bx + cx + d)$ and $\cos(ax + bx + cx + d)$. We can integrate those as follows. Let Ω be the triangle ABC . Let $\mathbf{k} = \langle a, b, c \rangle$ so that $ax + bx + cx = \mathbf{k} \cdot \mathbf{x}$. We then have

$$\alpha = \mathbf{k} \cdot (B - A)$$

$$\beta = \mathbf{k} \cdot (C - A)$$

$$\gamma = \mathbf{k} \cdot A + d$$

$$g(x, y) = \frac{\sin(x)/x - \sin(y)/y}{x - y}$$

$$h(x, y) = \frac{(1 - \cos(x))/x - (1 - \cos(y))/y}{x - y}$$

$$\frac{1}{\text{area}(\Omega)} \int_{\Omega} \sin(ax + by + cz + d) = 2(\sin(\gamma + \alpha)h(\beta - \alpha, -\alpha) - \cos(\gamma + \alpha)g(\beta - \alpha, -\alpha))$$

$$\frac{1}{\text{area}(\Omega)} \int_{\Omega} \cos(ax + by + cz + d) = 2(\cos(\gamma + \alpha)h(\beta - \alpha, -\alpha) + \sin(\gamma + \alpha)g(\beta - \alpha, -\alpha))$$

Although the functions $g(x, y)$ and $h(x, y)$ can be computed directly, care should be taken to avoid dividing by zero. Both functions are continuous and bounded everywhere.

3.5.3 Refinement Study

Next we consider interpolation over a MAC grid with irregular boundaries. We use a MAC grid on the domain $[0, 1]^3$. The domain is cut by two concentric spheres centered at $(0.5, 0.5)$ with radii 0.15 and 0.35, leaving a shell-shaped region, as illustrated in Figure 3.12. For this tests, we use the velocity fields $\mathbf{u}_{3a}$, $\mathbf{u}_{3c}$, $\mathbf{u}_{3d}$, and $\mathbf{u}_{3e}$. We populate the MAC grid from the analytic velocities as described in Section 3.5.2. Note that $\mathbf{u}_{3a}$ and $\mathbf{u}_{3d}$ are divergence-free while $\mathbf{u}_{3c}$ and $\mathbf{u}_{3e}$ are not.

$\mathbf{u}$	bc	Res	$\ell_2(\text{div})$	Order of error(ℓ_2)	$\ell_\infty(\text{div})$	Order of error(ℓ_∞)	$\ell_2(\text{err})$	Order of error(ℓ_2)	$\ell_\infty(\text{err})$	Order of error(ℓ_∞)	$\ell_\infty(\text{div}_a)$
$\mathbf{u}_{3a}$	Ave	8	3.77×10^{-8}	-	3.28×10^{-6}	-	6.31×10^{-3}	-	4.67×10^{-1}	-	2.65×10^{-13}
		16	3.10×10^{-8}	0.28	1.77×10^{-6}	0.89	1.39×10^{-3}	2.18	8.04×10^{-2}	2.54	9.50×10^{-13}
		32	3.60×10^{-8}	-0.22	2.14×10^{-5}	-3.60	3.91×10^{-4}	1.83	1.09×10^{-1}	-0.44	1.98×10^{-10}
		64	2.64×10^{-8}	0.45	1.30×10^{-5}	0.72	8.29×10^{-5}	2.24	3.32×10^{-2}	1.72	1.61×10^{-9}
		128	7.18×10^{-8}	-1.44	6.84×10^{-5}	-2.39	1.10×10^{-5}	2.91	2.58×10^{-3}	3.69	1.42×10^{-8}
		256	1.36×10^{-8}	2.41	1.42×10^{-6}	5.59	1.99×10^{-6}	2.47	2.57×10^{-4}	3.33	1.58×10^{-7}
		512	1.24×10^{-8}	0.13	6.77×10^{-7}	1.06	4.28×10^{-7}	2.22	2.95×10^{-5}	3.12	1.01×10^{-6}
$\mathbf{u}_{3c}$	Ave	8	1.98	-	2.37	-	2.87×10^{-3}	-	1.21×10^{-1}	-	2.37
		16	1.98	0.00	2.39	-0.01	7.51×10^{-4}	1.93	4.05×10^{-2}	1.58	2.39
		32	1.98	0.00	2.40	-0.00	1.90×10^{-4}	1.98	3.88×10^{-2}	0.06	2.40
		64	1.98	0.00	2.40	-0.00	4.11×10^{-5}	2.21	1.47×10^{-2}	1.40	2.40
		128	1.98	0.00	2.40	-0.00	6.91×10^{-6}	2.57	1.28×10^{-3}	3.52	2.40
		256	1.98	0.00	2.40	-0.00	1.40×10^{-6}	2.31	5.48×10^{-5}	4.55	2.40
		512	1.98	0.00	2.40	0.00	3.19×10^{-7}	2.13	1.03×10^{-5}	2.41	2.40
$\mathbf{u}_{3d}$	Ave	8	2.05×10^{-8}	-	9.77×10^{-7}	-	4.94×10^{-14}	-	2.64×10^{-12}	-	3.41×10^{-13}
		16	1.68×10^{-8}	0.29	8.99×10^{-7}	0.12	1.43×10^{-13}	-1.54	3.83×10^{-12}	-0.54	2.60×10^{-12}
		32	1.69×10^{-8}	-0.01	2.21×10^{-6}	-1.30	1.31×10^{-13}	0.13	3.97×10^{-12}	-0.05	4.96×10^{-10}
		64	1.78×10^{-8}	-0.07	1.15×10^{-6}	0.94	9.91×10^{-14}	0.40	6.30×10^{-12}	-0.66	3.96×10^{-9}
		128	1.85×10^{-8}	-0.05	7.11×10^{-7}	0.70	6.76×10^{-14}	0.55	2.78×10^{-12}	1.18	4.00×10^{-8}
		256	1.88×10^{-8}	-0.03	2.89×10^{-7}	1.30	5.13×10^{-14}	0.40	3.69×10^{-12}	-0.41	5.47×10^{-7}
		512	1.91×10^{-8}	-0.02	2.78×10^{-7}	0.06	3.47×10^{-14}	0.56	2.71×10^{-12}	0.44	1.71×10^{-6}
$\mathbf{u}_{3e}$	Ave	8	2.00	-	2.00	-	2.81×10^{-13}	-	1.26×10^{-11}	-	2.00
		16	2.00	0.00	2.00	0.00	6.12×10^{-13}	-1.12	1.61×10^{-11}	-0.35	2.00
		32	2.00	0.00	2.00	0.00	6.17×10^{-13}	-0.01	1.60×10^{-11}	0.01	2.00
		64	2.00	0.00	2.00	0.00	4.55×10^{-13}	0.44	3.91×10^{-11}	-1.29	2.00
		128	2.00	0.00	2.00	0.00	3.18×10^{-13}	0.52	1.05×10^{-11}	1.89	2.00
		256	2.00	0.00	2.00	-0.00	2.25×10^{-13}	0.50	1.42×10^{-11}	-0.43	2.00
		512	2.00	0.00	2.00	0.00	1.58×10^{-13}	0.51	1.25×10^{-11}	0.18	2.00

Table 3.2: Interpolation error on a shell-shaped domain for velocity fields $\mathbf{u}_{3a}$, $\mathbf{u}_{3c}$, $\mathbf{u}_{3d}$ and $\mathbf{u}_{3e}$ using the **full** interpolation basis. Fields $\mathbf{u}_{3a}$ and $\mathbf{u}_{3d}$ are divergence-free, while $\mathbf{u}_{3c}$ and $\mathbf{u}_{3e}$ are divergent; $\mathbf{u}_{3d}$ and $\mathbf{u}_{3e}$ are affine. “Ave” means that each grid value stores the integrated average of the analytic velocity over the corresponding face. The results show that our interpolation scheme reproduces affine fields exactly and interpolates divergence-free fields to be point-wise divergence-free. The converge rate is about two in the ℓ_2 norm, while the ℓ_∞ norm exhibits noisier convergence around order two. The last column reports the maximum absolute analytic divergence (the coefficient of the divergence basis mode), which is near zero up to round-off error when the underlying velocity field is divergence-free.

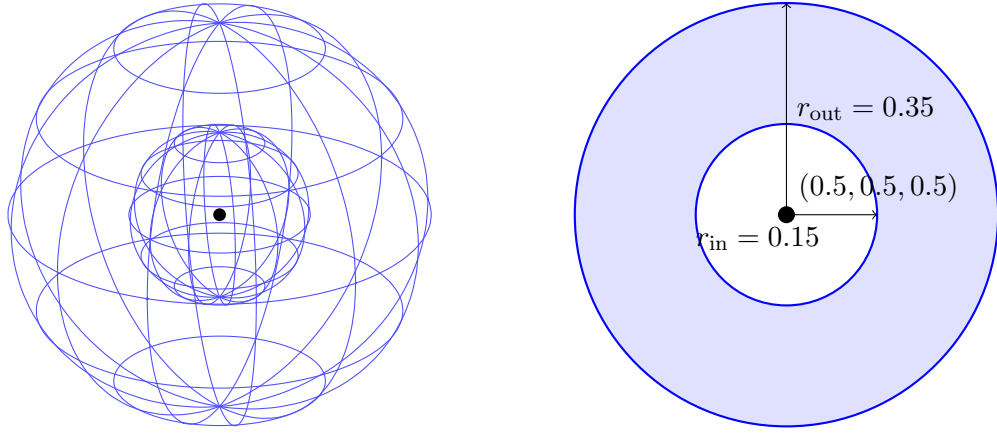


Figure 3.12: Spherical shell domain used in the refinement study. The shell is centered at $(0.5, 0.5, 0.5)$ within the unit cube, with inner radius $r_{\text{in}} = 0.15$ and outer radius $r_{\text{out}} = 0.35$. The left panel shows a three-dimensional view of the inner and outer bounding spheres. The right panel shows a cross-section through the center, with the shell region shaded and the center and two radii marked.

To evaluate the errors, we need to test errors at locations that are not grid locations. For this, we randomly seed 10^6 points, which we hold fixed across resolutions, ensuring that all of the points are inside the domain at all resolutions. As before, we estimate divergence at these points using finite differences as in (2.14). The ℓ_2 and ℓ_∞ errors are shown in Table 3.2. We observe second order in ℓ_2 for the nonlinear velocity fields $\mathbf{u}_{3a}$ and $\mathbf{u}_{3c}$. Errors in ℓ_∞ are much noisier, but the error follows the ℓ_2 error. The larger errors in ℓ_∞ are due to reduced accuracy in poorly conditioned tetrahedra; we investigate this further below.

We expect to see second order convergence, since our interpolation scheme recovers affine velocity fields but not quadratic velocity fields. $\mathbf{u}_{3d}$ and $\mathbf{u}_{3e}$ are affine velocity fields, so we expect them to be exactly represented by our interpolation scheme. We observe that the errors are close to roundoff levels at all resolutions and do not display any improvements in accuracy under refinement. Note that the magnitudes of the errors are about an order

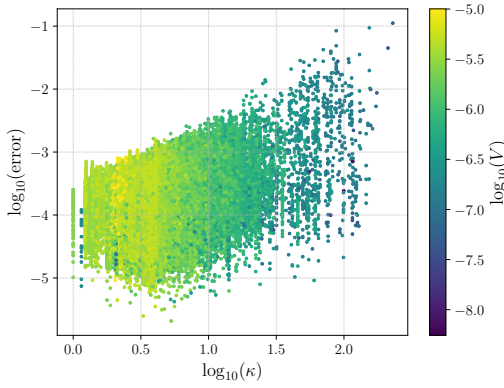
of magnitude larger when the underlying velocity field is not divergence-free. There are several steps in the algorithm designed to handle the complications of a divergent input; while these do succeed in recovering affine velocity fields, they also contribute significantly to the overall errors when recovering affine velocity fields.

One might expect the finite difference errors to be on the order of $\epsilon^2 = 10^{-12}$, or taking into account catastrophic cancellation, 10^{-10} . Even in $\ell_2(\text{div})$ we observe errors on the order of 10^{-8} . Nevertheless, these errors are due to the finite difference approximation used to evaluate the errors. Indeed, the errors $\ell_2(\text{div})$ and $\ell_\infty(\text{div})$ are similar between the two tests and across resolutions, even though the interpolation errors themselves are vastly different. This is consistent with these errors reflecting the accuracy of the finite difference divergence approximation itself rather than the interpolations scheme.

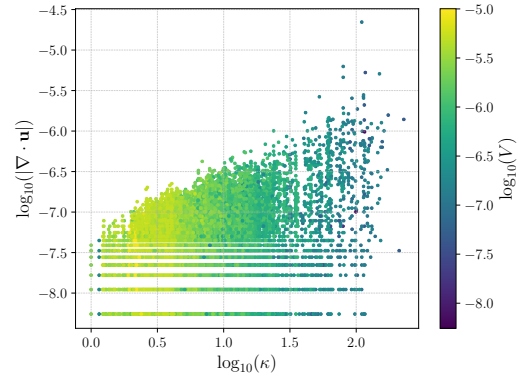
Because all of the basis functions are analytically divergence-free except for one divergent mode, the coefficient on that divergent mode provides another marker for the amount of divergence in the interpolated result. The maximum absolute value of this coefficient across all elements is shown in the final column $\ell_\infty(\text{div}_a)$. These coefficients are observed to be near roundoff errors at low resolution and increase with resolution. In the absence of roundoff errors, this coefficient would be exact at all resolutions. Its degradation reflects the tendency for sources of errors that affect its accuracy (such as poorly conditioned tetrahedra or catastrophic cancellation) to become more common at higher resolutions.

Next, we analyze the relationship between the errors we observe and the geometric properties of the tetrahedra. We use the same spherical shell geometry with $\mathbf{u}_{3a}$ at a resolution of 32. To validate the influence of tetrahedron conditioning on accuracy and divergence, we plot tetrahedron volume and condition number against interpolation errors

(Figure 3.13(a)), finite difference divergence errors (Figure 3.13(b)), and the size of the divergent mode's coefficient (Figure 3.14). We observe a clean correlation between how poorly conditioned a tetrahedron is and how large the each of the three errors for that tetrahedron can be. In particular, the larger errors observed in the l_∞ norm result from the poorly-conditioned tetrahedra that marching cubes occasionally produces. At the same time, these errors do not significantly affect the l_2 norm since such elements have very small volumes.



(a) Error versus tetrahedron condition number κ

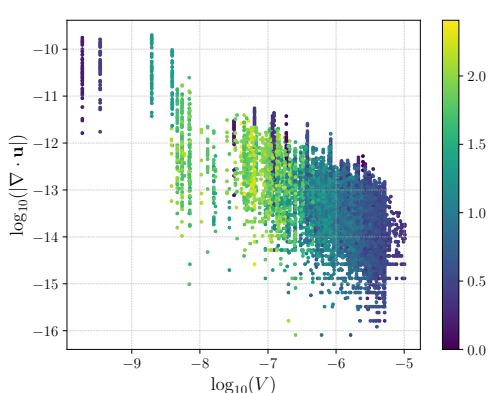


(b) Divergence versus tetrahedron condition number κ

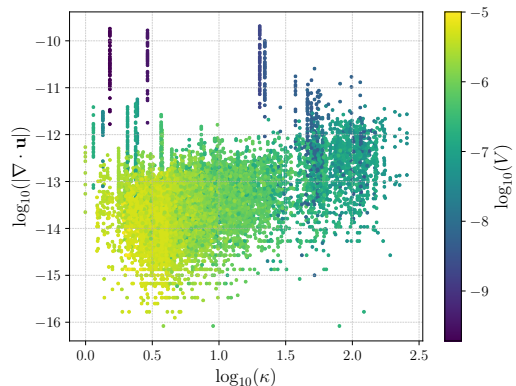
Figure 3.13: Approximation error and divergence as functions of element conditioning. In both cases the maximum observed error correlates strongly with condition number κ of the tetrahedron. This shows that errors seen in L_∞ arise from poorly conditioned elements and are sensitive to element conditioning. These poorly conditioned elements typically have very small volumes.

3.5.4 Full basis vs polynomial div-free basis

Our interpolation scheme constructs a full divergence-free basis on each element, which we supplement with a divergent mode. This allows us to be continuous across element boundaries. The cost we pay for this is the need to use rational functions for some of our



(a) Analytic divergence versus volume V



(b) Analytic divergence versus tetrahedron condition number κ

Figure 3.14: Analytic divergence (the coefficient of the divergence basis mode) plotted against element volume (left) and element condition number (right). The magnitude is at most 10^{-10} , effectively zero. The larger values correlate strongly with tetrahedron volume rather than with condition number.

basis functions. An alternative approach is to use a purely polynomial basis, such as Raviart-Thomas elements. We can emulate such a strategy within our framework by omitting the rational basis functions. This modification introduces a few subtle issues. The matrix $\mathbf{K}$ is no longer full rank, since we do not have a full basis. Simply using the pseudoinverse instead does not result in divergence-free interpolation. Instead, we select the component of the divergent mode so the remainder is divergence-free. Then, we use the pseudoinverse to map the divergence-free part onto the divergence-free modes. This procedure is linear and results in a new matrix $\hat{\mathbf{K}}$ that replaces $\mathbf{K}^{-1}$ in the original algorithm. If the basis had been complete, this procedure would recover the inverse.

We now repeat the refinement study from Section 3.5.3 using this strictly polynomial basis. The results are shown in Table 3.3 and are similar to what we observe in the original method. While the polynomial-only version may not be continuous at element

boundaries, it is nevertheless an accurate interpolation scheme. As such, we would also expect the discontinuities to converge to zero under refinement.

3.5.5 Path trace plotting

To demonstrate continuity across the various types of transitions that can occur between elements, we fill a MAC grid using our divergence-free but high frequency data $\mathbf{u}_{3b}$ as described in Section 3.5.2. This velocity field is not resolved on the grid and behaves as random data, though due to the integration procedure, this data is discretely divergence-free. We then plot a path through the domain as separate curves for u , v , and w . We use a coarse 16^3 domain and random data to make transitions easier to observe in the plots. Since the interpolation scheme is accurate, the discontinuities when interpolating well-resolved data would be too small to observe visually, though they could still cause problems for applications that rely on continuity. The path and results are shown in Figure 3.15. We observe that the curves for u , v , and w are C^0 continuous along the path. We also observe discontinuities in the slope, which we expect since the interpolation is not C^1 .

We note that in the interior, the random data does not appear very random and in fact is almost constant. Although the data is highly oscillatory, integrating it over a large MAC grid face cancels out most of the variation, resulting in MAC grid data that is nearly zero in the interior. Near the boundaries, cancellation is much less pronounced, resulting in much larger interpolated results in these areas. A similar phenomenon was observed in 2D [44], though averaging in two dimensions instead of one exacerbates the effect.

u	bc	Res	$\ell_2(\text{div})$	Order of error(ℓ_2)	$\ell_\infty(\text{div})$	Order of error(ℓ_∞)	$\ell_2(\text{err})$	Order of error(ℓ_2)	$\ell_\infty(\text{err})$	Order of error(ℓ_∞)	$\ell_\infty(\text{div}_a)$
u_{3a}	Ave	8	3.72×10^{-8}	-	3.31×10^{-6}	-	4.96×10^{-3}	-	2.79×10^{-1}	-	2.28×10^{-13}
		16	3.19×10^{-8}	0.22	2.38×10^{-6}	0.47	1.19×10^{-3}	2.06	4.90×10^{-2}	2.51	9.71×10^{-13}
		32	2.75×10^{-8}	0.21	2.75×10^{-6}	-0.21	2.89×10^{-4}	2.05	4.43×10^{-2}	0.15	1.99×10^{-10}
		64	2.01×10^{-8}	0.45	3.36×10^{-6}	-0.29	7.55×10^{-5}	1.93	5.35×10^{-2}	-0.27	1.62×10^{-9}
		128	1.61×10^{-8}	0.32	2.12×10^{-6}	0.67	9.99×10^{-6}	2.92	2.55×10^{-3}	4.39	1.42×10^{-8}
		256	1.36×10^{-8}	0.24	6.27×10^{-7}	1.76	1.92×10^{-6}	2.38	1.03×10^{-4}	4.62	1.58×10^{-7}
u_{3c}	Ave	512	1.25×10^{-8}	0.12	5.88×10^{-7}	0.09	4.26×10^{-7}	2.17	2.12×10^{-5}	2.29	1.01×10^{-6}
		8	1.98	-	2.37	-	2.41×10^{-3}	-	5.76×10^{-2}	-	2.37
		16	1.98	0.00	2.39	-0.01	6.79×10^{-4}	1.83	2.33×10^{-2}	1.31	2.39
		32	1.98	0.00	2.40	-0.00	1.60×10^{-4}	2.09	1.69×10^{-2}	0.46	2.40
		64	1.98	0.00	2.40	-0.00	3.92×10^{-5}	2.03	2.33×10^{-2}	-0.46	2.40
		128	1.98	0.00	2.40	-0.00	6.44×10^{-6}	2.61	1.02×10^{-3}	4.51	2.40
u_{3d}	Ave	256	1.98	0.00	2.40	-0.00	1.38×10^{-6}	2.22	8.08×10^{-5}	3.66	2.40
		512	1.98	0.00	2.40	0.00	3.21×10^{-7}	2.10	5.53×10^{-6}	3.87	2.40
		8	6.70×10^{-8}	-	3.69×10^{-6}	-	1.86×10^{-8}	-	1.04×10^{-6}	-	3.49×10^{-13}
		16	6.12×10^{-8}	0.13	2.83×10^{-6}	0.38	1.44×10^{-7}	-2.95	2.36×10^{-6}	-1.19	2.54×10^{-12}
		32	5.35×10^{-8}	0.19	8.66×10^{-6}	-1.62	1.38×10^{-7}	0.06	2.53×10^{-6}	-0.10	4.93×10^{-10}
		64	4.28×10^{-8}	0.32	1.75×10^{-5}	-1.02	1.04×10^{-7}	0.40	2.54×10^{-6}	-0.00	3.96×10^{-9}
u_{3e}	Ave	128	2.99×10^{-8}	0.51	4.05×10^{-6}	2.11	7.45×10^{-8}	0.48	2.38×10^{-6}	0.09	4.00×10^{-8}
		256	2.42×10^{-8}	0.31	2.23×10^{-6}	0.86	5.30×10^{-8}	0.49	2.33×10^{-6}	0.03	5.47×10^{-7}
		512	2.12×10^{-8}	0.19	7.11×10^{-7}	1.65	3.83×10^{-8}	0.47	2.11×10^{-6}	0.14	1.71×10^{-6}
		8	2.00	-	2.00	-	1.04×10^{-7}	-	6.06×10^{-6}	-	2.00
		16	2.00	0.00	2.00	0.00	1.01×10^{-6}	-3.29	1.15×10^{-5}	-0.93	2.00
		32	2.00	0.00	2.00	-0.00	9.55×10^{-7}	0.09	1.14×10^{-5}	0.02	2.00
u_{3e}	Ave	64	2.00	0.00	2.00	0.00	7.10×10^{-7}	0.43	1.25×10^{-5}	-0.14	2.00
		128	2.00	0.00	2.00	0.00	5.07×10^{-7}	0.49	1.21×10^{-5}	0.05	2.00
		256	2.00	0.00	2.00	-0.00	3.60×10^{-7}	0.49	1.18×10^{-5}	0.03	2.00
		512	2.00	0.00	2.00	0.00	2.58×10^{-7}	0.48	1.03×10^{-5}	0.20	2.00

Table 3.3: Interpolation error on a shell-shaped domain for velocity field $\mathbf{u}_{3a}$, $\mathbf{u}_{3c}$, $\mathbf{u}_{3d}$ and $\mathbf{u}_{3e}$ using only the **polynomial** interpolation basis. $\mathbf{u}_{3a}$ and $\mathbf{u}_{3d}$ are divergence-free, while the $\mathbf{u}_{3c}$ and $\mathbf{u}_{3e}$ are divergent. $\mathbf{u}_{3d}$ and $\mathbf{u}_{3e}$ are affine. “Ave” means that each grid value stores the integrated average of the analytic velocity over the corresponding face. The results confirm that our interpolation scheme preserves divergence-free for both $\mathbf{u}_{3a}$ and $\mathbf{u}_{3d}$. Disregarding the discontinuity across elements, the polynomial basis attains a convergence rate comparable to the full basis as the grid is refined.

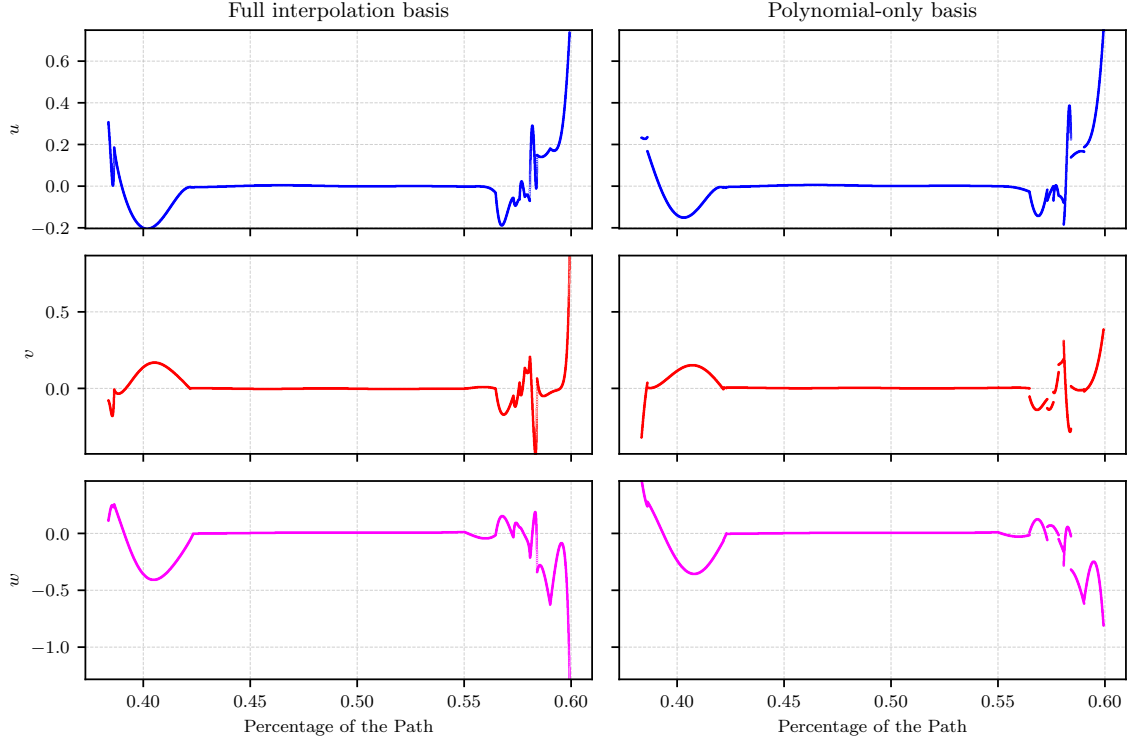


Figure 3.15: This figure shows the interpolated velocity along a path through the domain. The path passes through cut cells and transition cells near the ends of the path and through the interior cells near the middle. The input velocity field of the domain is $\mathbf{u}_{3b}$, which is divergence-free but sufficiently high frequency that it cannot be resolved on this grid. The left plot shows the interpolated velocity along the path when interpolating using the full bases in u (top), v (middle), and w (bottom). The interpolated velocity is continuous along the entire path. The right plot shows the interpolated velocity of the same path using only polynomial bases. This path has discontinuities as the path transitions from one element to the next. This lack of continuity is caused by the lack of a full interpolation basis, which prevents the interpolated field from matching the prescribed boundary values and produces a discontinuity. The use of an unresolved data causes some of these discontinuities to be large enough to be visually observed. When interpolating smooth data, these discontinuities would be too small to visually observe. On full uncut MAC grid faces, the integrated average causes the high-frequency by very regular velocity field to be averaged out to near zero, resulting in a nearly constant interpolation in the interior.

Polynomial basis only Next, we repeat the path trace using the polynomial-only basis, which we described in Section 3.5.4. The trace is shown in Figure 3.15. The general shape of the traces is similar to the case with the full basis. Examining the trace more closely, however, reveals several discontinuities in the interpolated profile. Thus, although a polynomial-only basis is as accurate as the full basis, it is unable to produce a continuous interpolation over the simple shapes. There are not enough degrees in the basis with only polynomials to match up with the element boundary conditions, so only a least-squares fit is achieved. Under refinement, these discontinuities will decrease as the errors in the interpolation decrease.

3.5.6 Random velocity field continuity test

In this test, we demonstrate compatibility of our divergence-free interpolation with non-analytical data. For our domain, we use a MAC grid with resolution 32^3 on the domain $[0, 1]^3$. We then cut this domain using a sphere with radius 0.3 centered at the center of the grid (0.5, 0.5, 0.5). We fill the MAC grid with random data in the range $[-1, 1]$. We set the boundary velocities to zero for compatibility. Next, we project the MAC grid values using the cut-cell projection discretization of Ng et al. [63], where we again enforce zero boundary conditions for compatibility. This produces MAC grid data that has been projected to be discretely divergence-free. The resulting velocity field has very high frequency, including deep in the interior. Note that although we specify zero velocities at the boundaries, the Poisson projection of Ng et al. [63] naturally enforces slip boundary conditions and so may introduce nonzero tangential components at the boundary.

As in Section 3.5.3, we seed the interior portion of the domain with 10^6 points. We ensure the particles lie inside the discretized region by excluding a thin band of width 10^{-3} (about 3% the width of a grid cell) along the analytical boundary. Following [44] we evaluate divergence at each point sample using (3.13) with $\epsilon = 10^{-8}$. The small stencil width reduces the chances of a finite difference stencil crossing element boundaries. Since the interpolation is C^0 but not C^1 , the derivative discontinuities would create large errors in the divergence approximation. The ℓ_2 and ℓ_∞ errors in the central-difference approximation of the divergence were 4.35×10^{-7} and 2.29×10^{-4} . In our interpolation scheme, all of the interpolation modes are divergence-free except for the one divergent mode $\langle x, y, z \rangle$. As before, we can use the coefficient for the divergent basis mode as another measure for the divergence in the reconstruction. The maximum absolute value of this coefficient over all the elements is 2.73×10^{-12} . These results are consistent with what was observed with individual elements.

3.5.7 Analytical velocity field projection test

Next, we perform a refinement study using Poisson projection on the well-resolved but non-divergence-free velocity field

$$\mathbf{u}' = \mathbf{u}_{3a} + \begin{pmatrix} (x^2 - x)(y^3/3 - y^2/2)(z^3/3 - z^2/2) \\ (y^2 - y)(x^3/3 - x^2/2)(z^3/3 - z^2/2) \\ (z^2 - z)(x^3/3 - x^2/2)(y^3/3 - y^2/2) \end{pmatrix}, \quad (3.14)$$

which is constructed as the sum of the divergence-free field $\mathbf{u}_{3a}$ and a term which is the gradient of a polynomial. The MAC grid data is computed from the analytic velocity as in

Section 3.5.2. We then project this velocity to be divergence-free using the projection scheme of Ng et al. [63], where we evaluate the boundary conditions from $\mathbf{u}_{3a}$. The projection should recover $\mathbf{u}_{3a}$ as the analytical solution. We use a **spherical shell** (Figure 3.12) for our grid domain, and we run the test over resolutions from 8 to 512. Let $\mathbf{w}$ be the projected velocity, which we can evaluate at arbitrary locations using our interpolation. The error is then $\mathbf{w} - \mathbf{u}_{3a}$. As in previous examples, we sample 10^6 points within the domain and use the equation (3.13) with $\epsilon = 10^{-8}$ to approximate the divergence. The results are shown in Table 3.4. The convergence orders are consistent with second order accuracy, which is consistent with the use of Ng et al. [63] and also the second order observed for our interpolation on an analytical velocity field. The divergence errors are fairly consistent across resolutions, since these errors are dominated by the central difference approximation, which is not changing under refinement.

3.6 Conclusion

In this paper, we have presented schemes for divergence-free interpolation in tetrahedra and cubes. We present the cube scheme for tensor product cubic polynomial boundary conditions, and we present the tetrahedron for quartic polynomials. This allows us to extend a spline-based divergence-free interpolation scheme to the boundary of an irregular domain. The resulting interpolation scheme is C^0 continuous and divergence-free.

$\mathbf{u}$	bc	Res	$\ell_2(\text{div})$	Order of error(ℓ_2)	$\ell_\infty(\text{div})$	Order of error(ℓ_∞)	$\ell_2(\text{err})$	Order of error(ℓ_2)	$\ell_\infty(\text{err})$	Order of error(ℓ_∞)	$\ell_\infty(\text{div}_a)$
$\mathbf{u}_{3a}$		8	3.81×10^{-8}	-	4.22×10^{-6}	-	6.31×10^{-3}	-	4.69×10^{-1}	-	1.98×10^{-13}
		16	3.11×10^{-8}	0.29	2.11×10^{-6}	1.00	1.39×10^{-3}	2.18	8.18×10^{-2}	2.52	9.72×10^{-13}
		32	3.67×10^{-8}	-0.24	2.16×10^{-5}	-3.36	3.90×10^{-4}	1.83	1.14×10^{-1}	-0.48	1.99×10^{-10}
		64	2.69×10^{-8}	0.45	1.28×10^{-5}	0.75	8.50×10^{-5}	2.20	3.42×10^{-2}	1.74	1.61×10^{-9}
		128	5.14×10^{-8}	-0.93	4.68×10^{-5}	-1.86	1.09×10^{-5}	2.97	2.65×10^{-3}	3.69	1.42×10^{-8}
		256	1.36×10^{-8}	1.92	1.10×10^{-6}	5.41	2.04×10^{-6}	2.41	2.42×10^{-4}	3.45	1.58×10^{-7}
		512	1.24×10^{-8}	0.13	5.22×10^{-7}	1.07	4.38×10^{-7}	2.22	2.68×10^{-5}	3.17	1.01×10^{-6}

Table 3.4: Results obtained by projecting a divergent velocity field using the projection scheme of [63]. Interpolating the projected data produces a velocity field that is analytically divergence-free. The reported divergence errors arise from the truncation error of the central-difference stencil used to evaluate the divergence, together with the effect of infrequent degenerate element cases; these errors do not decrease under refinement. The remaining errors measure the difference between the interpolated projected velocity field and the analytical projected solution. The observed convergence rate is approximately second order.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
u	1	z	z^2	z^3	z^4	y	yz	yz^2	yz^3	y^2	y^2z	y^2z^2	y^3	y^3z	y^4	0	0	0	0	0	$-x$
v	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	z	z^2	z^3	z^4	y
w	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36						
u	$-xz$	$-xz^2$	$-xz^3$	$-2xy$	$-2xyz$	$-2xyz^2$	$-3xy^2$	$-3xy^2z$	$-4xy^3$	0	0	0	0	$-x^2$	$-x^2z$						
v	yz	yz^2	yz^3	y^2	y^2z	y^2z^2	y^3	y^3z	y^4	x	xz	xz^2	xz^3	$2xy$	$2xyz$						
w	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0						
	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51						
u	$-x^2z^2$	$-x^2y$	$-x^2yz$	$-3x^2y^2$	0	0	0	$-x^3$	$-x^3z$	$-2x^3y$	0	0	$-x^4$	0	$-x$						
v	$2xyz^2$	xy^2	xy^2z	$2xy^3$	x^2	x^2z	x^2z^2	$3x^2y$	$3x^2yz$	$3x^2y^2$	x^3	x^3z	$4x^3y$	x^4	0						
w	0	0	0	0	0	0	0	0	0	0	0	0	0	0	z						
	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67						
u	$-2xz$	$-3xz^2$	$-4xz^3$	0	$-xy$	$-2xyz$	$-3xyz^2$	0	$-xy^2$	$-2xy^2z$	0	$-xy^3$	0	0	$-x^2$						
v	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0						
w	z^2	z^3	z^4	y	yz	yz^2	yz^3	y^2	y^2z	y^2z^2	y^3	y^3z	y^4	x	$2xz$						
	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82						
u	$-x^2z$	$-3x^2z^2$	0	$-x^2y$	$-x^2yz$	0	$-x^2y^2$	0	0	$-x^3$	$-2x^3z$	0	$-x^3y$	0	0						
v	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0						
w	xz^2	$2xz^3$	xy	$2xyz$	xyz^2	xy^2	$2xy^2z$	xy^3	x^2	$3x^2z$	$3x^2z^2$	x^2y	$3x^2yz$	x^2y^2	x^3						
				83	84	85	102														
u				$-x^4$	0	0	x														
v				0	0	0	y														
w				$4x^3z$	x^3y	x^4	z														

Table 3.5: Polynomial basis modes for tetrahedron. The last mode (102) is the divergent mode. All of the other modes are divergence-free. Although there are a lot of polynomial basis functions, they are very efficient to compute.

Table 3.6: Rational basis modes for tetrahedron. Note that quantities like $\frac{y}{x+y}$ and $\frac{x}{1-y-z}$ are bounded in $[0, 1]$, which allows them to be safely computed. The denominator can only go to zero on the canonical tetrahedron when the numerator also goes to zero. Note that A_{yzx} is the same as A_{xyz} but with the variables cycled.

	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101
u	A_{xyz}	0	B_{zxy}	C_{xyz}	D_{yzx}	0	0	E_{xyz}	F_{zxy}	G_{xyz}	I_{yzx}	H_{zxy}	L_{xyz}	O_{xyz}	Q_{yzx}	P_{zxy}
v	B_{xyz}	A_{yzx}	0	0	C_{yzx}	D_{zxy}	F_{xyz}	0	E_{yzx}	H_{xyz}	G_{yzx}	I_{zxy}	M_{xyz}	P_{xyz}	O_{yzx}	Q_{zxy}
w	0	B_{yzx}	A_{zxy}	D_{xyz}	0	C_{zxy}	E_{zxy}	F_{yzx}	0	I_{xyz}	H_{yzx}	G_{zxy}	N_{xyz}	Q_{xyz}	P_{yzx}	O_{zxy}

$$A_{xyz} = -\frac{z(z-1)^2(3x+2y)y^2}{(x+y)^2} \quad B_{xyz} = -\frac{y^3z(z-1)^2}{(x+y)^2} \quad C_{xyz} = -3\frac{y^2z(-1+y)^2}{1-x-y} \quad D_{xyz} = \frac{z^3y(-1+y)^2}{(1-x-y)^2}$$

$$E_{xyz} = \frac{x^3z(3yz+2z^2-2y-4z+2)}{(1-y-z)^2} \quad F_{xyz} = 3\frac{y^2z^2(-1+y)}{1-x-y} \quad G_{xyz} = -\frac{z(3z-2)(xy-3xz+y^2-2yz+3x+2y)^2}{(x+y)^2}$$

$$H_{xyz} = \frac{y^3z(3z-2)(z+x+y-1)}{(x+y)^2} \quad I_{xyz} = -3\frac{z^2(z-1)y^2}{x+y}$$

$$L_{xyz} = \frac{z^2}{(y+z)^2} \left(2\frac{z^3(z+1)^2}{(z+x)^2} - \frac{z^2(z+1)(9z+5)}{z+x} - \frac{z(z-1)^2y}{x+y} + 4z(x^2-xz+2z^2-2x+2z+1) \right) + \frac{z}{y+z} \left(\frac{z(z-1)^2y^2}{(x+y)^2} - \frac{z(z-1)^2y}{x+y} \right) - \frac{z^3(3y^3-4yz+2z^2-5y+4z+2)}{(z+x)^2} + \frac{z}{z+x} \left(-\frac{z(z-1)^2y^2}{(x+y)^2} + 2\frac{z(z-1)^2y}{x+y} + z(9y^3-16yz+9z^2-15y+14z+5) \right)$$

$$M_{xyz} = \frac{z^2}{(y+z)^2} \left(2\frac{z^3(z+1)^2}{(z+x)^2} - 4\frac{z^3(z+1)}{z+x} - z(x^3-2z^2-3x+2) \right) + \frac{z}{y+z} \left(-\frac{z^3(z+1)(9z+5)}{(z+x)^2} + 4\frac{z^3(6z+5)}{z+x} - \frac{z(z-1)^2y^2}{(x+y)^2} + z(3x^3+8xz-14z^2-9x-8z+6) \right) + \frac{z^2}{(z+x)^2} \left(\frac{z(z-1)^2y}{x+y} + z(4y^2-4yz+7z^2-8y+10z+3) \right) + \frac{z}{z+x} \left(\frac{z(z-1)^2y^2}{(x+y)^2} - \frac{z(z-1)^2y}{x+y} - 4z(3y^2-4yz+5z^2-3y+4z) \right)$$

Table 3.6: (continued)

$$\begin{aligned}
N_{xyz} &= \frac{z^2}{(y+z)^2} \left(-2 \frac{z^3(z+1)^2}{(z+x)^2} + 4 \frac{z^3(z+1)}{z+x} + z(x^3 - 2z^2 - 3x + 2) \right) + \frac{z^3(3y^3 - 4yz + 2z^2 - 5y + 4z + 2)}{(z+x)^2} + 4 \frac{z^3(2y - z - 1)}{z+x} \\
O_{xyz} &= \frac{x^2}{(1-y-z)^2} \left(2 \frac{z^2(z-1)y^2}{(x+y)^2} - 2 \frac{z(7z-4)(z-1)y}{x+y} + 4z(3z-2)(z-1) \right) + 4 \frac{y^2 z(3z-2)}{x+y} \\
&\quad + \frac{x}{1-y-z} \left(-2 \frac{z(z-1)(z-2)y^2}{(x+y)^2} - 8 \frac{z^2(z-1)y}{x+y} + 2z(6xz + 5z^2 - 4x - 7z + 2) \right) \\
P_{xyz} &= \frac{x^2}{(1-y-z)^2} \left(\frac{z^2(z-1)y}{1-x-z} - 2 \frac{z^2(z-1)y^2}{(x+y)^2} + \frac{z^2(z-1)y}{x+y} \right) \\
&\quad + \frac{x}{1-y-z} \left(\frac{z^2(z-1)y^2}{(1-x-z)^2} - 2 \frac{(2z-1)z(z-1)y}{1-x-z} + 2 \frac{z(z-1)(z-2)y^2}{(x+y)^2} + \frac{z(17z-14)(z-1)y}{x+y} - 16z(z-1)^2 \right) \\
&\quad + \frac{y^2}{(1-x-z)^2} \left(\frac{z^2(z-1)y}{x+y} - z^2(z-1) \right) + \frac{y}{1-x-z} \left(\frac{z(3z-2)(z-1)y}{x+y} + z(3z-2)(z-1) \right) - 4 \frac{y^2 z(3z-2)}{x+y} + 10yz(3z-2) \\
Q_{xyz} &= \frac{x^2}{(1-y-z)^2} \left(-\frac{z^2(z-1)y}{1-x-z} + 2 \frac{z^2(z-1)y^2}{(x+y)^2} - \frac{z^2(z-1)y}{x+y} \right) \\
&\quad + \frac{x}{1-y-z} \left(-\frac{z^2(z-1)y}{1-x-z} - 3 \frac{z^2(z-1)y^2}{(x+y)^2} + 12 \frac{z^2(z-1)y}{x+y} - 8z^2(z-1) \right) + \frac{y}{1-x-z} \left(\frac{z^2(z-1)y^2}{(x+y)^2} - 3 \frac{z^2(z-1)y}{x+y} + 2z^2(z-1) \right)
\end{aligned}$$

Table 3.7: Polynomial basis modes for cube. Basis mode 168 is the divergent mode; the rest are divergence-free. As with the tetrahedron, there are a lot of polynomial basis modes, but they are simple and can be very efficiently computed.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
u	0	0	0	0	0	0	1	y	z	0	x	0	0	0	0	0	0	0	0	0	0	yz
v	0	0	0	1	x	z	0	0	0	y	$-y$	0	0	0	0	0	xz	x^2	x^3	z^2	z^3	0
w	1	x	y	0	0	0	0	0	0	$-z$	0	xy	x^2	x^3	y^2	y^3	0	0	0	0	0	0
	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40				
u	y^2	y^3	z^2	z^3	0	0	0	0	0	0	0	0	yz^2	yz^3	y^2z	y^3z	0	xy				
v	0	0	0	0	0	0	0	0	xz^2	xz^3	x^2z	x^3z	0	0	0	0	xy	0				
w	0	0	0	0	xy^2	xy^3	x^2y	x^3y	0	0	0	0	0	0	0	0	$-xz$	$-yz$				
	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56						
u	xz	0	0	0	0	0	0	y^2z^2	y^2z^3	y^3z^2	0	0	y^3z^3	0	0	x^2						
v	$-yz$	0	0	0	x^2z^2	x^2z^3	x^3z^2	0	0	0	0	0	x^3z^3	0	$2yz$	y^2	$-2xy$					
w	0	x^2y^2	x^2y^3	x^3y^2	0	0	0	0	0	0	x^3y^3	0	0	$-z^2$	$-2yz$	0						
	57	58	59	60	61	62	63	64	65	66	67	68	69	70								
u	0	0	0	0	0	$3xy^2$	$3xz^2$	xy^3	xz^3	x^2y	x^2z	x^3	0	0								
v	$3yz^2$	x^2y	x^3y	y^2z	y^3	$-y^3$	0	0	$-yz^3$	$-xy^2$	0	$-3x^2y$	$2xyz$	xy^2								
w	$-z^3$	$-x^2z$	$-x^3z$	$-yz^2$	$-3y^2z$	0	$-z^3$	$-y^3z$	0	0	$-xz^2$	0	$-xz^2$	$-2xyz$								
	71	72	73	74	75	76	77	78	79	80	81	82	83									
u	$2xyz$	0	0	0	0	$3xyz^2$	$3xy^2z$	x^2yz	x^3y	x^3z	x^3y^2	x^3z	0									
v	$-y^2z$	$3xyz^2$	xy^2z	xy^3	y^3z^2	0	$-y^3z$	$-xy^2z$	0	$-x^2y^3$	$-3x^2yz$	0	$2x^2yz$									
w	0	$-xz^3$	$-xyz^2$	$-xyz^2$	$-y^2z^3$	$-yz^3$	0	0	$-3x^2yz$	0	0	$-x^2z^3$	$-x^2z^2$									
	84	85	86	87	88	89	90	91	92	93	94	95										
u	0	0	0	0	0	$2xyz^3$	$2xy^3z$	$3x^2y^2$	$3xz^2$	x^2y^3	x^2z^3	0										
v	$2x^3yz$	$2y^3z$	$3y^2z^2$	x^2y^2	x^3y^2	$-y^2z^3$	0	$-2xy^3$	0	0	$-2xyz^3$	$3x^2yz^2$	0									
w	$-x^3z^2$	$-3y^2z^2$	$-2yz^3$	$-2x^2yz$	$-2x^3yz$	0	$-y^3z^2$	0	$-2xz^3$	$-2xy^3z$	0	$-2xz^3$	$-2xy^3z$	0								

Table 3.7: (continued)

	96	97	98	99	100	101	102	103	104	105	106
u	0	0	0	0	0	0	$3xy^2z^2$	$3xy^2z^3$	$3xy^3z^2$	$x^2y^2z^2$	x^2yz^3
v	$3x^3yz^2$	xy^3z^2	x^2y^2z	x^2y^3	x^3y^2z	x^3y^3	$-y^3z^2$	$-y^3z^3$	0	$-xy^2z^2$	$-xy^2z^3$
w	$-x^3z^3$	$-xy^2z^3$	$-x^2yz^2$	$-3x^2y^2z$	$-x^3yz^2$	$-3x^3y^2z$	0	0	$-y^3z^3$	0	0
	107	108	109	110	111	112	113	114	115	116	117
u	x^2y^2z	x^2y^3z	x^3yz^2	x^3y^2z	x^3y^3	x^3z^3	0	0	$2x^3yz$	0	0
v	0	0	0	$-x^2y^3z$	0	$-3x^2yz^3$	$2xy^3z$	$3xy^2z^2$	$-3x^2y^2z$	$x^2y^3z^2$	$x^3y^3z^2$
w	$-xy^2z^2$	$-xy^3z^2$	$-x^2yz^3$	0	$-3x^2y^3z$	0	$-3xy^2z^2$	$-2xy^3z$	0	$-x^2y^2z^3$	$-x^3y^2z^3$
	118	119	120	121	122	123	124	125	126	127	
u	$x^3y^2z^2$	$x^3y^2z^3$	$x^3y^3z^2$	0	0	0	0	$2x^3yz^3$	$2x^3y^3z$	$3x^2y^2z^2$	
v	$-x^2y^3z^2$	$-x^2y^3z^3$	0	$2x^2y^3z$	$2x^3y^3z$	$3x^2y^2z^2$	$3x^3y^2z^2$	$-3x^2y^2z^3$	0	$-2xy^3z^2$	
w	0	0	$-x^2y^3z^3$	$-3x^2y^2z^2$	$-3x^3y^2z^2$	$-2x^2yz^3$	$-2x^3yz^3$	0	$-3x^2y^3z^2$	0	
	128	129	130	131	132						
u	$3x^2y^2z^3$	$3x^2y^3z^2$	$-2x^2y^3z(z-1)(2x-3)$	$3x^4yz(z-1)(y-1)$	$-2x^2yz^3(y-1)(2x-3)$						
v	$-2xy^3z^3$	0	$3xzy^4(z-1)(x-1)$	$-2x^3zy^2(z-1)(2y-3)$	0						
w	0	$-2xy^3z^3$	0	0	$3xyz^4(y-1)(x-1)$						
	133	134	135	136							
u	$-3x^4yz^2(z-1)(y-1)$	$-3x^4y^2z(z-1)(y-1)$	$2x^2y^3z^2(z-1)(2x-3)$	x							
v	$2x^3y^2z^2(z-1)(2y-3)$	0	$-3xz^2y^4(z-1)(x-1)$	y							
w	0	$2x^3y^2z^2(2z-3)(y-1)$	0	z							

Table 3.8: Rational basis modes for cube. The variables r , s , t and intermediates f_{xy} are defined for symmetry and simplicity. The quantities $f_{xy} \in [0, 1]$ can be implemented robustly. These modes are highly symmetrical, which makes them manageable to implement.

	136	137	138	139	140	141	142	143	144	145	146	147
u	A_{xyz}	0	B_{zxy}	$-A_{ryz}$	$-B_{zry}$	A_{xsx}	0	$-A_{rsz}$	0	B_{txy}	$-B_{try}$	0
v	B_{xyz}	A_{yzx}	0	B_{ryz}	0	$-B_{xsx}$	$-A_{szx}$	$-B_{rsz}$	A_{ytx}	0	0	$-A_{stx}$
w	0	B_{yzx}	A_{zxy}	0	A_{zry}	0	B_{sxx}	0	$-B_{ytx}$	$-A_{txy}$	$-A_{try}$	$-B_{stx}$
	148	149	150	151	152	153	154	155	156	157	158	159
u	C_{xyz}	0	D_{zxy}	$-C_{ryz}$	$-D_{zry}$	C_{xsx}	0	$-C_{rsz}$	0	D_{txy}	$-D_{try}$	0
v	D_{xyz}	C_{yzx}	0	D_{ryz}	0	$-D_{xsx}$	$-C_{szx}$	$-D_{rsz}$	C_{ytx}	0	0	$-C_{stx}$
w	0	D_{yzx}	C_{zxy}	0	C_{zry}	0	D_{sxx}	0	$-D_{ytx}$	$-C_{txy}$	$-C_{try}$	$-D_{stx}$
	160	161	162	163	164	165	166	167				
u	E_{xyz}	$-E_{ryz}$	E_{xsx}	$-E_{rsz}$	E_{xyt}	$-E_{ryt}$	E_{xst}	$-E_{rst}$				
v	F_{xyz}	F_{ryz}	$-F_{xsx}$	$-F_{rsz}$	F_{xyt}	F_{ryt}	$-F_{xst}$	$-F_{rst}$				
w	G_{xyz}	G_{ryz}	G_{xsx}	G_{rsz}	$-G_{xyt}$	$-G_{ryt}$	$-G_{xst}$	$-G_{rst}$				

$$f_{xy} = \frac{x}{x+y} \quad r = 1-x \quad s = 1-x \quad t = 1-x$$

$$A_{xyz} = -z(y-1)(x-1)^2(z-1)(4xy+3y^2-2x-y)f_{xy}f_{yx}$$

$$B_{xyz} = z(y-1)^2(x-1)(z-1)(2x^2+3xy-y)f_{yx}^2$$

Table 3.8: (continued)

$$\begin{aligned}
C_{xyz} &= -z(1-x)^2(z-1)^2(y-1)(4xy+3y^2-2x-y)f_{xy}f_{yx} \\
D_{xyz} &= -z(y-1)^2(z-1)^2(1-x)(2x^2+3xy-y)f_{yx}^2 \\
E_{xyz} &= z(x-1)^2(z-1)(y-1)[(2(y-1)(y+1)f_{zy}^2+4f_{zy}y^2+4xy+2)f_{zx}^2+((-11y^2+2y+5)f_{zy}^2-y(17y-7)f_{zy} \\
&\quad + (y-1)^2f_{xy}^2+3y(y-1)f_{xy}+16x^2-16xy+16xz+4x+4z-6)f_{zx}-12(x+y)f_{xy} \\
&\quad + ((y-1)(y+1)f_{xy}+6xy+9y^2-2x-2y-3)f_{zy}^2+(y^2-1)f_{xy}^2+(2y^2+y+1)f_{xy}+13y^2-6x-7y)f_{zy}] \\
F_{xyz} &= -z(z-1)^2(y-1)^2(x-1)[+(2f_{zy}^2y-3f_{zy}y-(y+1)f_{xy}^2+(2y+1)f_{xy})f_{zx}+(-6x-4y+2)f_{zy}^2 \\
&\quad ((2y-2)f_{zy}^2-(3y-5)f_{zy}-4+(y+1)f_{xy})f_{zx}^2+((y+1)f_{xy}^2-(3y+2)f_{xy}+18x+6y-5)f_{zy}+12(x+y)f_{xy}] \\
G_{xyz} &= 2z(z-1)^2(y-1)(x-1)[((y-1)^2f_{zy}^2+2y(y-1)f_{zy}+2xy-1)f_{zx}^2+(y(y-1)f_{zy}^2+y(2y-1)f_{zy}+4xy)f_{zx} \\
&\quad - (y-1)(3x+2y-1)f_{zy}^2-y(3x+4y-3)f_{zy}]
\end{aligned}$$

Although we only present the basis functions for the degrees of polynomials that we required for our final reconstruction, the basis is readily adapted to other degrees. Reducing the polynomial degree is straightforward, since the new basis will be a linear combination of the existing one. The size of the basis required shrinks dramatically with the required polynomial degree, resulting in a more efficient interpolation for applications that can use lower polynomial degree.

Our scheme inherits some limitations of the corresponding 2D scheme. The rational basis functions are Lipschitz continuous, but they are not differentiable at the edges or corners of the shape, and the method does not explicitly optimize the conditioning of the tetrahedra. Our extension into the gap region also relies on the interior spline-based interpolation scheme being flux-consistent. Relaxing this assumption should be possible, and would be an interesting direction for future exploration.

Our scheme requires tetrahedra to have non-negative volume and to match up with neighboring tetrahedra. For this, we construct a special marching cubes table that is able to produce a tetrahedralization of the interior whose tetrahedra have non-negative volumes and match continuously between grid cells. Marching cubes tables adapted to produce tetrahedra appear occasionally (such as for computing finite element integrals over cut cells), but the properties of the resulting tetrahedra are typically left unspecified. Non-inverted tetrahedra and continuity between elements are not needed for computing integrals. Given the difficulty of obtaining those properties without adding additional interior vertices, it seems likely that such tables lack one or both properties. Our table is designed to provide both.

Chapter 4

Reducing the Boiling Effect in the Material Point Method through Consistent Particle Initialization

4.1 Introduction

The Material Point Method (MPM) [\[80\]](#) is a numerical method that simulates materials using two structures at the same time: a set of particles and a background grid. Each particle carries the material properties, such as mass, volume, velocity, and deformation gradient. At every time step, the particle information is transferred from the particles to the grid, the equations of motion are solved on the grid, and the result is transferred back to update the particles. Because the particles move freely while the grid stays fixed, MPM handles large deformation naturally, without the mesh distortion problems of pure

mesh-based methods [14]. For this reason, MPM has been widely used to simulate snow [79], sand [47, 24] and other complex materials in both computer graphics [92, 87] and computational mechanics [73, 25].

The accuracy of MPM depends on how well the particles perform this transfer. The governing equations require integrals over the material domain, such as the internal force acting on each grid node. MPM approximates this integral by *quadrature*: the integral is replaced by a weighted sum over the particles, where each particle acts as a sample point and its volume acts as the weight. For example, the internal force at a grid node i is computed as

$$\mathbf{f}_i = - \int_{\Omega} \sigma \nabla N_i d\mathbf{x} \approx - \sum_p \sigma_p \nabla N_i(\mathbf{x}_p) V_p, \quad (4.1)$$

where σ_p is the particle stress, N_i is the grid basis function, $\mathbf{x}_p$ is the particle position, and V_p is the particle volume. This approximation is accurate only when the particle positions and volumes are consistent with the grid: the particles must cover the domain evenly, and each volume must match the space the particle actually represents. When particles are clustered or leave gaps, or when their volumes do not add up correctly under the grid basis functions, the weighted sum misrepresents the true integral. This gap between the exact integral and the particle sum is *quadrature error*, and the analysis of Steffen et al. [76], Baumgarten and Kamrin [13] showed that it can influence the overall accuracy of MPM.

Quadrature error is especially harmful at the very start of a simulation. Consider a tank of water at rest under gravity: physically, the internal forces should exactly balance gravity, and nothing should move. However, if the initial particle state is inconsistent, the quadrature in Eq. (4.1) evaluates the force integrals inaccurately, and the grid receives forces

that do not balance. The particles then begin to move even though no physical cause exists. This spurious motion ranges from persistent noisy jittering to coherent, directed drift, and is often strongest near the domain boundary; because the disturbed surface resembles boiling water, we refer to it as the *boiling effect*. The inconsistency has two sources. In the interior, the particle positions and volumes must be consistent with the background grid, so that the particle volumes correctly tile space under the grid basis functions. Near the boundary, the particles must additionally be consistent with the shape of the domain itself: where the boundary cuts through grid cells, particles tend to claim too much or too little volume, so the quadrature effectively integrates over the wrong region. In both cases, the problem does not come from the material model or the time integrator: it is already present in the initial particle state, before the first time step is taken. Common sampling strategies such as regular lattices with jittering or Poisson disk sampling [19] produce distributions that look uniform to the eye, but satisfy neither consistency requirement, and therefore still trigger the boiling effect.

Particle disorder is not unique to MPM. Smoothed Particle Hydrodynamics (SPH) [60], a fully particle-based method with no background grid, suffers from a closely related problem: its kernel approximations are accurate only when the particles are evenly distributed, and as particles move during the simulation, this uniformity is gradually destroyed. The SPH community developed a family of *runtime* corrections that repair the distributions during time stepping. Particle shifting techniques [88, 51] move each particle a small distance toward regions of lower particle concentration every time step, keeping the distribution close to uniform. Diffusive schemes such as δ -SPH [56] take a different route: instead of moving particles, they add a small diffusion term to the density equation that smooths out the noise

caused by the disorder. Both approaches are effective and widely used. And both treat the symptoms continuously, applying a correction at every time step for the entire simulation, rather than addressing the inconsistency appearing at the beginning of the simulation.

In MPM, runtime remedies mainly improve the transfer between particles and grid. One line of work replaces the point-sample quadrature with richer particle representation. For example, the Generalized Interpolation Material Point method (GIMP) [11] uses smoother representations of discrete material point data and treats each particle as a small block rather than a point. The Convected Particle Domain Interpolation (CPDI) [68] further lets this block deform with the material. Both smooth the transfers and reduce the error a particle causes when it crosses a cell boundary. A second line of work improves what is transferred: the Affine Particle-In-Cell (APIC) method [42] and the Moving Least Squares material point method (MLS-MPM) [40] carry a locally linear velocity field on each particle, making the transfers more accurate and less dissipative. Closer in spirit to SPH shifting, Zhang et al. [94] detect regions where the particle density has drifted from its correct value during the simulation and shift particles along the density gradient to correct it. Baumgarten and Kamrin [13] propose the δ -correction, which shifts material points incrementally during a simulation to counteract flow-induced clumping. All of these techniques operate during time stepping, and they do improve the overall accuracy and stability of MPM. However, they don't explicitly address the inconsistency that exists at the start of the simulation. If the initial positions and volumes are already inconsistent with the grid and the domain, the boiling effect can still occur due to the spurious startup forces.

A complementary line of work fixes the particle distribution *before* the simulation starts. In SPH, this is known as particle packing or particle relaxation. The basic recipe

was introduced by Colagrossi et al. [23]. The idea is starting from a rough initial sampling, an auxiliary simulation is run in which particles repel each other and gradually settle, and this fictitious dynamics is iterated until the particles reach a steady, evenly spaced configuration; the real simulation then starts from the settled state. The quality target of this process is the kernel consistency condition, $\sum_j V_j \nabla_i W_{ij} \approx \mathbf{0}$, where W_{ij} is the smoothing kernel between particles i and j and the gradient is taken with respect to the position of particle i ; the condition states that the kernel gradients around each particle should cancel out, as they would for a perfectly uniform distribution. Litvinov et al. [52] showed that while particles perturbed off a uniform grid lose zero-order consistency and exhibit non-decaying integration error, such relaxed distributions restore the partition-of-unity property, recovering convergence rates comparable to those of particles on a uniform grid. Later work extended the recipe to practical geometry. Negi and Ramachandran [62] developed algorithms for domains with complex boundaries and provided an open-source implementation. Boregowda and Liu [17] extended packing to boundary-integral SPH, incorporating the boundary integral terms to the shifting dynamics so that particles conform to surface-element boundaries without ghost-particle layers. Moreover, Zhao et al. [96] extended relaxation to multi-body systems, coupling the relaxation of contacting bodies so that zero-order consistency improves at fluid-solid interfaces. This improvement is critical for Eulerian SPH, where particles never rearrange after initialization.

These initialization methods do not carry over directly to MPM, for three reasons. First, they target the different consistency condition. All of them drive the zero-order condition $\sum_j V_j \nabla_i W_{ij} \approx \mathbf{0}$ (the kernel gradient summation [96]), or, through a diffusion analogy, the concentration gradient [17], which is intrinsic to SPH, where particles interact

with their neighbors through kernels. MPM particles interact only with the grid, and the relevant condition is that the quadrature in Eq. (4.1) reproduces the true integrals under the grid basis functions. A distribution satisfying the kernel condition can still be inconsistent with a particular grid. Second, they correct by simulation rather than by solving. Pseudo-dynamic relaxation iterates a fictitious dynamics and waits for equilibrium: uniformity indicators can be monitored, but the iteration is not a descent method for any objective, the indicators can stall or rise mid-run [17], and termination falls back to heuristic such as fixed iteration counts or relative-change thresholds [96]. Third, particle volume is never treated as an unknown. Relaxation moves positions only. However, in MPM, particle volume V_p enters the quadrature directly as the weight of each sample, which could be adjusted as well. Within MPM itself, the closest relative is the δ -correction of Baumgarten and Kamrin [13]. Clumping is detected via a nodal “over-shoot” measure comparing the particle-integrated volume against the exact nodal volume, and particles are shifted by a gradient descent step on this error norm; the measure is one-sided, detecting over-integration but not under-sampling. The method therefore shares the position-only, correct-by-nudging character of the SPH relaxation methods, and it operates at runtime rather than on the initial state. An initialization method formulated for MPM itself, one that measures the full two-sided grid residual and corrects both particle volumes and particle positions by explicitly minimizing it, is still missing.

In this work, we address the boiling effect at its source: we correct the initial particle state itself, before the first time step is taken. We formulate the inconsistency as an explicit objective, the residual between the particle-rasterized nodal volumes and the nodal volumes implied by the domain discretization itself. The residual can be driven down

through either set of unknowns of the quadrature, giving two solvers: a linear *volume correction* solved as a regularized least-squares system, and a nonlinear *position redistribution* solved with a Gauss–Newton method, which cannot produce invalid particle volumes and therefore serves as our main approach. The correction runs once, before the simulation starts, and requires no modification of the MPM time stepping itself.

4.2 Discretization

4.2.1 Background

The Material Point Method (MPM) is a hybrid particle–grid discretization. The material is represented by a set of Lagrangian particles that carry all persistent state variables, such as mass, volume, velocity, and deformation history. A fixed background Eulerian grid is used as a computational scratchpad on which the momentum equation is solved each time step. The core pipeline of each time step is a particle-to-grid transfer of mass and momentum, a grid-based solve of the momentum equation, and a grid-to-particle transfer that updates particle velocities and positions.

Particles are indexed by p , with positions $\mathbf{x}_p \in \Omega \subset \mathbb{R}^d$ ($d = 2, 3$), volumes V_p , masses $m_p = \rho_0 V_p$ (where ρ_0 is the rest density and Ω the initial material domain), velocities $\mathbf{v}_p$, and deformation gradients $\mathbf{F}_p$. The grid is uniform with spacing h , and its nodes are indexed by i with positions $\mathbf{x}_i$. Particles and grid communicate through interpolation weights $w_{ip} = N_i(\mathbf{x}_p)$ and their gradients $\nabla w_{ip} = \nabla N_i(\mathbf{x}_p)$, where N_i is the basis function of node i , defined below.

We describe the general pipeline following the formulation of [79]; concrete implementation, including the one used in this work, may differ in individual stages for practical reasons (e.g., choice of basis functions, time integration scheme, choice of transfer scheme, or constitutive update) without affecting the structure relevant to this chapter. Each time step consists of three stages.

Particle-to-grid transfer (P2G). Mass and momentum are gathered from the particles onto the grid nodes:

$$m_i = \sum_p w_{ip} m_p, \quad (m\mathbf{v})_i = \sum_p w_{ip} m_p \mathbf{v}_p, \quad \mathbf{v}_i = \frac{(m\mathbf{v})_i}{m_i}. \quad (4.2)$$

Grid update. Internal forces are computed at the nodes from the particle stresses. Then Cauchy stress $\boldsymbol{\sigma}_p = \boldsymbol{\sigma}(\mathbf{F}_p)$ of each particle p is evaluated from its deformation gradient $\mathbf{F}_p$ through the constitutive model. The nodal force is then

$$\mathbf{f}_i = - \sum_p V_p \boldsymbol{\sigma}_p \nabla w_{ip} + m_i \mathbf{g}, \quad (4.3)$$

and the grid velocities are advanced,

$$\mathbf{v}_i^{n+1} = \mathbf{v}_i^n + \Delta t \frac{\mathbf{f}_i}{m_i}, \quad (4.4)$$

followed by the enforcement of boundary conditions on the grid velocities.

Grid-to-particle transfer (G2P). The updated grid velocities are interpolated back to the particles, the particles are advected, and the deformation gradient is updated from the grid velocity gradient:

$$\mathbf{v}_p^{n+1} = \sum_i w_{ip} \mathbf{v}_i^{n+1}, \quad \mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t \mathbf{v}_p^{n+1}, \quad \mathbf{F}_p^{n+1} = \left(\mathbf{I} + \Delta t \sum_i \mathbf{v}_i^{n+1} (\nabla w_{ip})^\top \right) \mathbf{F}_p^n. \quad (4.5)$$

The interpolation kernel. Throughout this work, N_i is the tensor-product quadratic B-spline

$$N_i(\mathbf{x}) = \prod_{k=1}^d N\left(\frac{x^{(k)} - x_i^{(k)}}{h}\right), \quad N(u) = \begin{cases} \frac{3}{4} - u^2, & |u| \leq \frac{1}{2}, \\ \frac{1}{2} \left(\frac{3}{2} - |u|\right)^2, & \frac{1}{2} < |u| \leq \frac{3}{2}, \\ 0, & \text{otherwise,} \end{cases} \quad (4.6)$$

which is C^1 . And that force (4.3), which involves ∇w_{ip} , remains continuous as particles cross cell boundaries and has compact support $[-\frac{3}{2}h, \frac{3}{2}h]^d$ centered at the node, so that each particle interacts with at most 3^d nodes.

4.2.2 The Boiling Problem

Consider the standard initialization pipeline. Particles are sampled inside the material domain Ω by Poisson-disk sampling, with an approximately uniform density of n_{ppc} particles per grid cell, and every particle is assigned the same volume,

$$V_p = \frac{|\Omega|}{N_p} \approx \frac{h^d}{n_{ppc}}, \quad (4.7)$$

where N_p is the total particle count. This choice appears natural: the particle volumes tile the domain exactly, and the sampling is unbiased. Nevertheless, a simulation initialized this way exhibits a persistent artifact in the simplest possible setting. A body of fluid at rest under gravity, which should remain exactly quiescent, develops grid-scale velocity noise from the first time step and never settles; the particles jitter and churn indefinitely, as if the fluid were boiling (Figure 4.1). We refer to this artifact as the *boiling problem* or *boiling effect*.

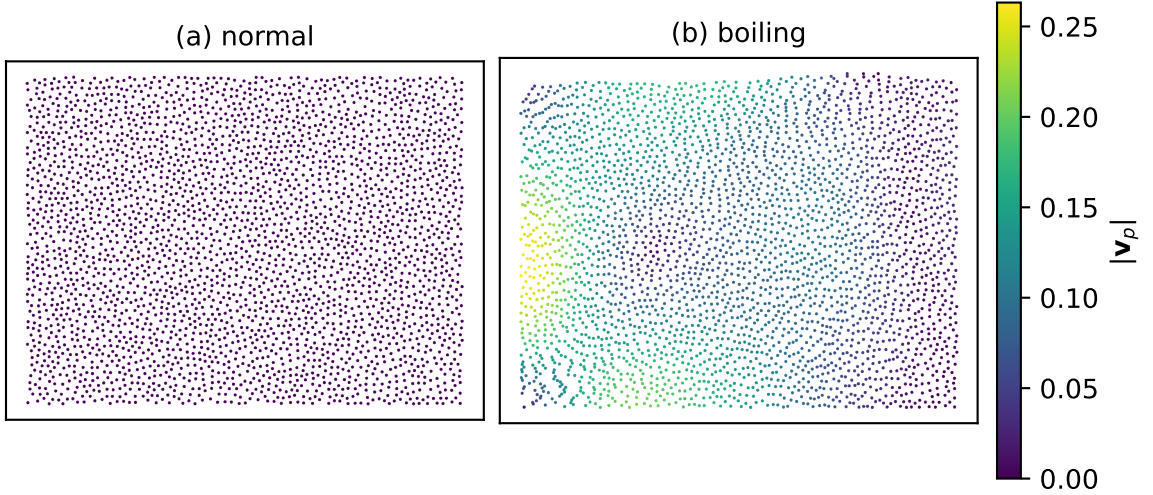


Figure 4.1: The boiling effect. A body of fluid at rest under gravity, whose exact solution is $\mathbf{v} = \mathbf{0}$ for all time, is simulated with the natural particle initialization. (a) The initial state, at rest. (b) The boiling effect: the particles exhibit spurious velocities, colored by magnitude $|\mathbf{v}_p|$. The agitation persists.

We first observe what does *not* go wrong. The weights satisfy the partition of unity, $\sum_i N_i(\mathbf{x}) = 1$, so summing the rasterized nodal volume $u_i = \sum_p w_{ip} V_p$ over all nodes gives

$$\sum_i u_i = \sum_i \sum_p V_p w_{ip} = \sum_p V_p \sum_i w_{ip} = \sum_p V_p = |\Omega|, \quad (4.8)$$

for any particle arrangement, and the same argument applied to the mass gather in (4.2) shows that the total mass is likewise exact. Global conservation holds by construction, and yet the artifact appears: the defect must be local.

Each node should receive a share of volume equal to the material it represents. In practice it receives $u_i = \sum_p w_{ip} V_p$, a weighted sum over the particles in its support. The discrepancy between the target geometric volume and the volume actually received depends on the sample distribution. Poisson-disk sampling is uniform only on average: one node's support holds a few more particles, another's a few fewer. Since every particle carries the same volume, u_i inherits these fluctuations directly. The error is typically worse near the domain boundary, where a node represents only a fraction of the support volume.

The nodewise mismatch enters the dynamics through the internal-force quadrature (4.3). For a fluid at rest under gravity, the continuum internal force balances gravity at every interior node. The discrete sums do not balance. The gravity force $m_i \mathbf{g}$ inherits the w_{ip} -weighted mass gather of (4.2), while the internal force (4.3) is a quadrature weighted by ∇w_{ip} . When the particles are irregularly distributed, the two errors do not cancel. The residual becomes spurious nodal forces and, through (4.4), spurious velocities in the absence of any physical disturbance. Eliminating the artifact therefore requires matching the gathered volume to the node's representative volume node by node. And making this precise requires defining what a node's representative volume is, which is the subject of the next section.

4.2.3 Proposed Solution: Volume Matching

We define the volume that node i represents as the domain integral of its weight function,

$$b_i = \int_{\Omega} N_i(\mathbf{x}) d\mathbf{x}, \quad (4.9)$$

called the *target volume*. For a node whose support lies inside Ω , $b_i = h^d$. For nodes near the boundary, including nodes outside Ω whose support reaches in, b_i lies partial between 0 and h^d . Evaluating (4.9) is the subject of Section 4.3.

The proposed initialization requires the rasterized volume to match the target at every node:

$$u_i = \sum_p w_{ip} V_p = b_i \quad \text{for all } i. \quad (4.10)$$

The effect on the density estimate is immediate. If the particles are initialized with a uniform density ρ_0 , then the density represented by node i is

$$\rho_i = \frac{\sum_p w_{ip} m_p}{b_i} = \rho_0 \frac{u_i}{b_i}, \quad (4.11)$$

Thus, under (4.10), every node represents exactly the density ρ_0 .

The condition (4.10) defines a system with two sets of unknowns: the particle volumes V_p , for which the residual $r_i = u_i - b_i$ depend linearly on the unknowns, and the particle positions $\mathbf{x}_p$, for which the dependence is nonlinear through $w_{ip} = N_i(\mathbf{x}_p)$. The next section develops solvers for each set of unknowns separately.

4.2.4 Solving the Matching Problem

The matching condition (4.10) can be pursued through either set of unknowns. We first correct the particle volumes, which leads to a linear solve. Its shortcoming then motivates the main approach of this work, which corrects the particle positions.

Volume Correction

We first hold the particle positions fixed and treat the particle volumes as the unknowns. Let n_n denote the number of nodes and n_p denote the number of particles. We define $W \in \mathbb{R}^{n_n \times n_p}$ by $W_{ip} = w_{ip}$.

The matching condition then becomes the linear system

$$W\mathbf{V} = \mathbf{b}.$$

Since $n_p \gg n_n$, this system is underdetermined and generally admits many solutions. To select a unique solution, we penalize deviations from the seed particle volumes $\mathbf{V}^0$ defined in (4.7). We therefore minimize

$$E(\mathbf{V}) = \frac{1}{2} \|W\mathbf{V} - \mathbf{b}\|^2 + \frac{\lambda}{2} \|\mathbf{V} - \mathbf{V}^0\|^2, \quad (4.12)$$

where $\lambda > 0$ controls the strength of the regularization. Setting the gradient to zero yields the normal equations

$$(W^\top W + \lambda I) \mathbf{V} = W^\top \mathbf{b} + \lambda \mathbf{V}^0. \quad (4.13)$$

We write this system in the standard form

$$A\mathbf{V} = \mathbf{f},$$

where

$$A = W^\top W + \lambda I, \quad \mathbf{f} = W^\top \mathbf{b} + \lambda \mathbf{V}^0.$$

Since $\lambda > 0$, the matrix A is symmetric positive definite. Therefore, the linear system (4.13) is well suited to the conjugate gradient (CG) method. As a Krylov-subspace method, CG accesses A only through matrix-vector products. It can therefore be implemented without assembling or factorizing the system matrix. Start from an initial guess $\mathbf{V}_0$, CG updates the solution as

$$\mathbf{V}_{k+1} = \mathbf{V}_k + \alpha_k \mathbf{q}_k,$$

where $\mathbf{q}_k$ is the search direction at iteration k and α_k is the corresponding step length. To compute α_k and compute the residual, CG requires the matrix-vector product $A\mathbf{q}_k$. In this problem, that product can be evaluated without assembling A :

$$A\mathbf{q}_k = W^\top (W\mathbf{q}_k) + \lambda \mathbf{q}_k.$$

The product $W\mathbf{q}_k$ gathers particle values to the nodes using the weights w_{ip} . The product $W^\top \mathbf{y}$ then spreads the resulting nodal vector $\mathbf{y}$ back to the particles through the same weights. Thus each iteration requires one particle-to-grid pass and one grid-to-particle

pass. Neither W nor the much larger matrix $W^\top W$ is stored. This matrix-free formulation makes the solve practical even for systems with millions of particles. Each CG iteration requires two passes over the particles, and the convergence is determined by a residual tolerance. Since each iteration steps along a best descent direction of the quadratic (4.12), the objective is sufficiently reduced within a modest budget, typically 200 iterations in our experiments.

The method has one structural flaw. Nothing in (4.12) constrains the volumes to remain strictly positive, and in practice the correction does drive some particle volumes negative, particularly near the boundary where the target volumes are small. A particle with negative volume carries negative mass and is physically meaningless. This flaw motivates the main approach of this work: leave the volumes untouched and move particles instead.

Position Correction

We now hold the particle volumes fixed at the seed values $\mathbf{V}^0$, which are positive by construction, and treat the particle positions as the unknowns. Let $\mathbf{X} \in \mathbb{R}^{dn_p}$ collect all particle positions, and let $\mathbf{X}^0$ denote the initial positions produced by the Poisson-disk sampling. The residual of the nodal volume-matching condition, condition,

$$\mathbf{r}(\mathbf{X}) = W(\mathbf{X}) \mathbf{V}^0 - \mathbf{b},$$

is now a nonlinear function of $\mathbf{X}$, since the weights $w_{ip} = N_i(\mathbf{x}_p)$ depend on the positions. As in the volume correction, we regularize the solution toward the initial sample. Here, the regularization penalizes particle displacements, and we minimize

$$E(\mathbf{X}) = \frac{1}{2} \|\mathbf{r}(\mathbf{X})\|^2 + \frac{\lambda}{2} \|\mathbf{X} - \mathbf{X}^0\|^2, \quad (4.14)$$

where $\lambda > 0$ keeps the corrected positions close to the sampled distribution, preserving its blue-noise quality rather than allowing particles to drift arbitrarily far in pursuit of the fit. Unlike particle volumes, the position variables are not subject to a positivity constraint. We can therefore use a relatively small regularization parameter, allowing the solver to correct the nodal volume mismatch while introducing only modest particle displacements.

Since E is a smooth nonlinear function, we minimize it with Newton's method. Newton's method replaces E near the current iterate $\mathbf{X}_k$ by its second-order Taylor model and steps toward the minimizer of that model. Let

$$\mathbf{g}_k = \nabla E(\mathbf{X}_k), \quad H_k = \nabla^2 E(\mathbf{X}_k)$$

denote the gradient and Hessian at $\mathbf{X}_k$. The Newton direction $\boldsymbol{\delta}_k$ is obtained by solving

$$H_k \boldsymbol{\delta}_k = -\mathbf{g}_k. \quad (4.15)$$

The iterate is then updated according to

$$\mathbf{X}_{k+1} = \mathbf{X}_k + \alpha_k \boldsymbol{\delta}_k,$$

where α_k is a step length chosen to ensure sufficient decrease of the objective. Newton's method is attractive for its rapid local convergence. Near a nondegenerate minimizer, the second-order model becomes accurate and the iterates converge quadratically. As a result, only a small number of Newton iterations is typically required.

The objective (4.14), the gradient and Hessian have a special structure inherited from the least-squares form. Let $J = \partial \mathbf{r} / \partial \mathbf{X}$ denote the Jacobian of the residual. Its entry coupling node i to particle p is

$$J_{ip} = \frac{\partial r_i}{\partial \mathbf{x}_p} = V_p^0 \nabla N_i(\mathbf{x}_p)^\top,$$

a $1 \times d$ block built from the weight gradient. Then

$$\mathbf{g} = J^\top \mathbf{r} + \lambda(\mathbf{X} - \mathbf{X}^0), \quad H = J^\top J + \sum_i r_i \nabla^2 r_i + \lambda I.$$

The Gauss-Newton method approximates the Hessian by dropping the middle term:

$$H \approx J^\top J + \lambda I. \tag{4.16}$$

This approximation has three common advantages. It requires no second derivatives of the weights. The dropped term is proportional to the residual, so the approximation is accurate when the residual is small, which is exactly the regime the iteration drives toward. Most importantly, the approximate Hessian (4.16) is symmetric positive definite by construction, so every step is a descent direction, whereas the full Hessian can become indefinite when the

residual does not vanish. In our experiments the full Newton and Gauss–Newton Variants produced very similar convergence behavior, and we therefore use Gauss–Newton.

The complete procedure is summarized as follows. Line 4 is where the Krylov

Algorithm 1 Position correction by Gauss–Newton

```

1:  $\mathbf{X}_0 \leftarrow \mathbf{X}^0$  ▷ Poisson-disk sample
2: for  $k = 0, 1, 2, \dots$  until  $\|\mathbf{g}_k\| < \varepsilon$  do
3:   compute  $\mathbf{r}(\mathbf{X}_k)$ ,  $\mathbf{g}_k$ 
4:   solve  $(J^\top J + \lambda I)\boldsymbol{\delta}_k = -\mathbf{g}_k$  by CG
5:   find  $\alpha_k$  by Wolfe line search
6:    $\mathbf{X}_{k+1} \leftarrow \mathbf{X}_k + \alpha_k \boldsymbol{\delta}_k$ 
7:   project particles outside  $\Omega$  back to the surface
8: end for

```

solver enters. Each Gauss–Newton step is a symmetric positive definite linear system of the same kind as (4.13), so it is solved by the same matrix-free CG method, with the operator $J^\top J + \lambda I$ and the right-hand side $-\mathbf{g}_k$ passed to the solver. As before, CG only needs the matrix–vector product: multiplying by J is a particle-to-grid pass and multiplying by $J^\top$ is a grid-to-particle pass, now using the weight gradients $\nabla N_i(\mathbf{x}_p)$ in place of the weights. Neither J nor $J^\top J$ is stored, so each Newton iteration remains as cheap as a few transfer passes over the particles.

The implementation details complete the method. First, the step length α_k in line 5 is chosen by a line search satisfying the Wolfe conditions, which guarantees sufficient decrease of E at every iteration. Second, a step may move a particle outside the domain; in line 7 such particles are projected back onto the domain surface along the normal direction of the level set, so that all particles remain inside Ω throughout the iteration. The outer iteration runs for at most ten Newton steps, or fewer if the gradient norm falls below a

tolerance. In practice, ten iterations reduce the residual enough for the volume match to serve its purpose.

The position correction avoids the main limitation of volume correction. Each particle retains its positive seed volume V_p^0 , so negative volumes and masses cannot occur. Before the correction, particle volumes are globally rescaled by the ratio of the total target node volume to the total particle volume, so that the correct total volume is achievable regardless of the number of seeded particles. The resulting particle state satisfies the nodal volume matching condition while remaining physically valid. For this reason, position correction is the primary approach used in this work.

4.3 Evaluating the Target Volume

This section presents three techniques for evaluating the target volume (4.9), in the order they were developed: a closed-form approximation based on a radially symmetric kernel, an exact computation for box domains, and a quadrature method for arbitrary geometry. Each was motivated by a limitation of the previous one, as we describe along the way.

Radially Symmetric Approximation

The target volume (4.9) is the kernel integral over the portion of its support inside the domain. Near a flat boundary, this region is obtained by cutting the support with a plane, as illustrated in Figure 4.2. For a radially symmetric kernel, the integral depends only on the signed distance d from the node to the plane. It is independent of the plane orientation. This motivates replacing the tensor-product B-spline with a radial kernel,

reducing the target volume to a closed-form function $f(d)$ evaluated at each node's level-set value.

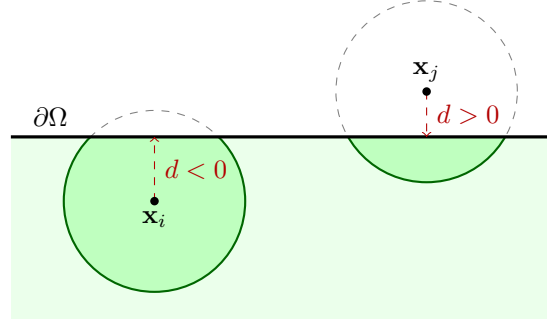


Figure 4.2: Signed distance to a flat boundary. An interior node $\mathbf{x}_i$ ($d < 0$) retains most of its kernel support inside the domain, while an exterior node $\mathbf{x}_j$ ($d > 0$) contributes only a small spherical cap. In both cases the clipped integral is the same closed-form function $f(d)$, independent of the plane's orientation.

We choose the form

$$\hat{N}(r) = A \left(1 - \frac{r^2}{R^2}\right)^2, \quad r \leq R, \quad (4.17)$$

which is radially symmetric, polynomial in r like the quadratic B-spline, and compactly supported with $R = \frac{3}{2}h$, The largest radius inscribed in the B-spline support. We work at $h = 1$, so $R = \frac{3}{2}$; general h is recovered by scaling at the end. The constant A is fixed by one constraint: a node deep in the interior must receive the same target as under the true kernel, namely the unit cell volume. Since the kernel is used only for volume computation and not for transfers, this matching of the total integral is the property that matters, and no partition of unity is required. In 2D the constraint reads

$$\int_0^R A \left(1 - \frac{r^2}{R^2}\right)^2 2\pi r dr = 1 \quad \implies \quad A = \frac{4}{3\pi},$$

and in 3D

$$\int_0^R A \left(1 - \frac{r^2}{R^2}\right)^2 4\pi r^2 dr = 1 \implies A = \frac{35}{36\pi}.$$

We now derive $f(d)$. Place the node at the origin and the boundary at $y = -d$, so that the domain is the half-plane $\{y \leq -d\}$: for $d < 0$ the node lies inside the domain and the boundary passes above the origin, and for $d > 0$ below. The target is the weighted volume of the kernel support below this line,

$$f(d) = \int_{\{y \leq -d\} \cap \{r \leq R\}} \hat{N}(r) d\mathbf{x}.$$

In 2D, integrating over x first reduces the inner integral to a closed form in y , and the outer integral evaluates to

$$f(d) = \frac{(-256 d^5 + 1872 d^3 - 5346 d) \sqrt{9 - 4d^2} + 10935 \arccos\left(\frac{2d}{3}\right)}{10935 \pi}, \quad (4.18)$$

In 3D, the result is the simpler degree-seven polynomial

$$f(d) = \frac{1}{2} - \frac{35}{48} d + \frac{35}{108} d^3 - \frac{7}{81} d^5 + \frac{20}{2187} d^7. \quad (4.19)$$

Both hold on $d \in [-\frac{3}{2}, \frac{3}{2}]$, with $f \equiv 1$ below this range and $f \equiv 0$ above. For general grid spacing, the volume is computed using the scaled kernel, giving

$$f_h(d) = h^2 f_1\left(\frac{d}{h}\right) \text{ in 2D,} \quad f_h(d) = h^3 f_1\left(\frac{d}{h}\right) \text{ in 3D,} \quad (4.20)$$

where f_1 is the unit formula above. In practice, each node evaluates f_h at its level-set value and the target is assigned directly.

We verified the approximation on smoothly curved boundaries by comparing it against direct numerical integration of the tensor-product B-spline over a circular domain of radius $10h$ in 2D and a spherical domain of radius $4h$ in 3D. All grid nodes whose kernel support intersects the boundary were evaluated: since the support is a cube of half-width $\frac{3}{2}h$, this band extends to node distances of $\frac{3}{2}\sqrt{2}h$ and $\frac{3}{2}\sqrt{3}h$ from the boundary in 2D and 3D respectively. Reference values were computed by Gauss–Legendre quadrature of the B-spline over its support inside the domain. Figure 4.3 reports the error as a function of the signed distance to the domain boundary. The absolute error remains below 3.9% of the unit cell volume in 2D and 5.5% in 3D, concentrated near the boundary and vanishing toward the interior, so the approximation is adequate where the boundary is smooth.

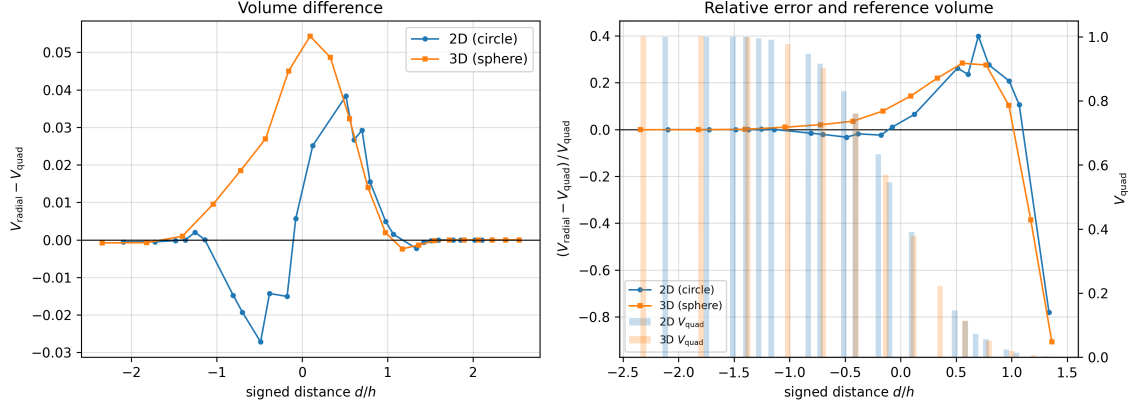


Figure 4.3: Error of the radially symmetric closed form on a circle of radius $10h$ (2D) and a sphere of radius $4h$ (3D), against Gauss–Legendre quadrature of the tensor-product B-spline. Volumes are normalized by the unit cell volume, so an interior node has $V_{\text{quad}} = 1$. Left: volume difference versus signed distance. Right: relative error (lines, left axis) over nodes with reference volume above 10^{-3} , with the reference volume shown as bars (right axis). The largest relative errors occur where the reference volume is nearly zero, so their absolute contribution is small; the drop toward -100% past $d = \frac{3}{2}h$ reflects the spherical support assumed by the closed form.

The relative error in Figure 4.3, however, already exposes the structural limits of the single-distance form, which stem from two geometric simplifications. First, the closed form assumes a spherical support, but the true support is a cube. Beyond $|d| = \frac{3}{2}h$ the formula is already zero while the cube corners still overlap the domain, so the relative error reaches -100% . Second, a single signed distance cannot encode boundary curvature, visible as the positive bias at moderate distances, much less two intersecting walls. Both effects are mild on smooth domains. At the edges and corners of a box, the support is cut by two or three planes at once, and the single-wall formula overestimates the target by up to a factor of two and four, respectively. In our 2D experiments the approximation suppresses the boiling effectively, but in 3D the residual boiling concentrates precisely along the wall–wall edges. This observation motivated the exact computation of the next section.

Exact Evaluation for Box Domains

To confirm that the residual boiling in 3D is caused by the target error at edges and corners, we compute the exact targets for an axis-aligned box domain. Two structural facts make this possible. The kernel is a tensor product, and the box is a product of intervals, so the integral (4.9) factorizes into one-dimensional integrals along each axis:

$$b_i = \int_{\Omega} N_i(\mathbf{x}) d\mathbf{x} = \prod_{k=1}^d \int_{a_k}^{c_k} N\left(\frac{x^{(k)} - x_i^{(k)}}{h}\right) dx^{(k)}, \quad (4.21)$$

where $[a_k, c_k]$ is the extent of the box along axis k . We assume the walls of the box coincide with grid lines. With the quadratic B-spline supported on three cells, only a few cases arise. Integrating the kernel (4.6) over the interval gives the exact fractions

$$\frac{1}{h} \int N = \begin{cases} \frac{1}{48}, & \text{node one cell outside the wall,} \\ \frac{1}{2}, & \text{node on the wall,} \\ \frac{47}{48}, & \text{node one cell inside the wall,} \\ 1, & \text{node two or more cells inside,} \end{cases} \quad (4.22)$$

and zero for nodes two or more cells outside. The exact target of node i is then the product of its per-axis fractions,

$$b_i = h^d \prod_{k=1}^d f_k(i), \quad (4.23)$$

where $f_k(i)$ is the fraction (4.22) selected by the node's integer distance to the nearer wall on axis k . The assignment is a lookup with no integration at run time.

The product structure quantitatively explains why the radial approximation fails at wall-wall edges and corners. Replacing the approximate targets by the exact values (4.23) eliminates the residual boiling in our 3D experiments, confirming that the target accuracy at these nodes governs the artifact. The exact computation, however, is tied to the box geometry; general domains require the method of the next section.

Quadrature Evaluation for General Domains

The exact computation of the previous section confirms that accurate targets eliminate the artifact, but it relies on the factorization over a box. For a general domain, no closed form is available, and we evaluate the integral (4.9) by numerical quadrature.

The method requires no new machinery. We place a regular lattice of quadrature points over the grid, with q points per axis in each cell, and discard the points outside the domain by testing the level set. Each remaining point represents an equal share of the cell volume, $\Delta V_q = h^d/q^d$. The target of node i is then the quadrature sum

$$b_i \approx \sum_{q \in \Omega} N_i(\mathbf{x}_q) \Delta V_q, \quad (4.24)$$

which is a particle-to-grid gather over the quadrature points, with the points in place of the particles and their volume shares in place of the particle volumes. The implementation reuses the transfer infrastructure of Section 4.2.1 directly, treating the lattice as a temporary particle set, and the cost is a single gather pass before the simulation starts. This construction mirrors the problem it solves. The boiling arose because the simulation particles form a poor quadrature rule for the integrals (4.9); the targets are computed by

the same gather, but over a regular sample with correct weights. An example is shown in Figure 4.4.

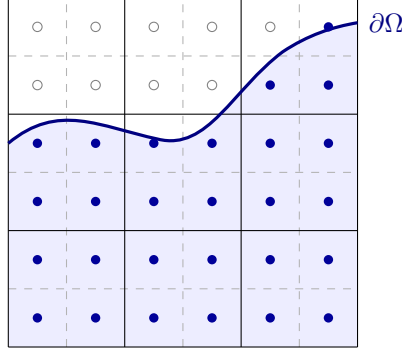


Figure 4.4: Quadrature evaluation of the target volumes. A regular lattice places $q \times q$ points per grid cell (here $q = 2$; solid lines are grid cells of size h , dashed lines the quadrature subcells). Points outside the domain (open circles) are discarded by the level-set test; each remaining point (filled) carries an equal volume share $\Delta V_q = h^d/q^d$, and the targets are computed by the standard particle-to-grid gather over this set.

The parameter q controls the accuracy. In our experiments the q can be a reasonable small value (e.g., $q = 2$ or $q = 3$) for the box domain, and the resulting simulations behave identically. The method extends to any domain with a level set, at any boundary orientation, and requires neither the flat-boundary assumption of the radial approximation nor the box structure of the exact computation.

4.4 Numerical tests

4.4.1 Setup

All experiments in this section use a hydrostatic test: water fills the bottom portion of a unit box (a square in 2D, a cube in 3D) under gravity $g = 9.8$, with the walls of the box coinciding with the domain boundary. The fluid is weakly compressible with a simple

pressure model, and the exact solution is trivial: the water remains at rest, so any particle velocity is artifact. Particles are seeded by Poisson-disk sampling at the default density per cell, and the grid resolution is 32 cells per axis unless stated otherwise. Each run is advanced for 100 frames.

We report two scalar summaries of the maximum particle speed, $(v_{\max}(t_k))$, over a 100-frame run: the peak speed,

$$v_{\text{peak}} = \max_k v_{\max}(t_k),$$

and the RMS speed,

$$v_{\text{rms}} = \left(\frac{1}{100} \sum_k v_{\max}(t_k)^2 \right)^{1/2}.$$

The peak captures the worst instantaneous artifact, while the RMS reflects its persistence. Sustained boiling keeps $(v_{\max}(t_k))$ high across many frames, whereas a brief transient affects only a small number of frames.

4.4.2 Parameter Settings

The position correction of Section 4.2.4 has two parameters: the regularization weight λ and the Newton stopping tolerance. Used naively, both would require retuning at every resolution, because the two terms of the objective (4.14) shrink at different rates under grid refinement: the volume residuals scale with the cell volume, so the residual term scales as h^{2d} , while the displacements scale with the cell size, so the penalty scales as h^2 . We remove this dependence by nondimensionalizing the objective, dividing volumes by h^d and measuring displacements in cells, which makes both terms pure numbers. The dimensionless

weight $\hat{\lambda}$ and the tolerance on the dimensionless gradient are then tuned once, on a 2D run at resolution 32, and frozen for all experiments in this section, across resolutions and across 2D and 3D. We use $\hat{\lambda} = 10^{-6}$ and a gradient tolerance of 1×10^{-6} , with at most ten Newton iterations and at most 200 CG iterations per step, as described in Section 4.2.4.

4.4.3 Comparison of Target Evaluation Methods

Tables 4.1 and 4.2 compare the three target node volume evaluation techniques of Section 4.3 on the hydrostatic test in 2D and 3D respectively, all using the position correction with the parameters of Section 4.4.2. The uncorrected baseline is included for reference. To check that the conclusions do not depend on a particular configuration, each technique is run at three water levels (0.25, 0.5, and 0.75 of the box height) and under two wall boundary conditions, slip and separating. For each run we report v_{peak} and v_{rms} .

In 2D (Table 4.1), the radially symmetric approximation, exact evaluation, and quadrature-based evaluation all significantly reduce both the peak and RMS velocities compared to the uncorrected baseline. The effect is most pronounced at water level 0.75, where the baseline yields $v_{\text{peak}} = 0.26$ and $v_{\text{rms}} = 0.21$. Exact evaluation and quadrature-based evaluation reduce v_{peak} to 0.06 and v_{rms} to 0.018, while the radially symmetric approximation achieves a comparable but slightly weaker reduction, to 0.07 and 0.02 respectively. The performance is similar under both wall boundary conditions, slip and separating. The reduction in velocities is likewise consistent across all water levels, although the baseline itself degrades with increasing water level, exhibiting more pronounced boiling effects. For

Table 4.1: v_{peak} and v_{rms} over 100 frames of the 2D hydrostatic test at resolution 32, for three water levels and two wall boundary conditions.

BC	Target evaluation	Water level					
		0.25		0.5		0.75	
		v_{peak}	v_{rms}	v_{peak}	v_{rms}	v_{peak}	v_{rms}
slip	None (baseline)	0.0465	0.0215	0.127	0.101	0.263	0.211
	Radial approximation	0.0308	0.00645	0.0516	0.0135	0.069	0.0199
	Exact (box)	0.0181	0.00437	0.0364	0.0105	0.0642	0.0181
	Quadrature ($q = 2$)	0.018	0.00439	0.0363	0.0105	0.0639	0.0181
	Quadrature ($q = 3$)	0.018	0.00438	0.0363	0.0105	0.0641	0.0181
sep	None (baseline)	0.0462	0.0217	0.127	0.101	0.264	0.212
	Radial approximation	0.0307	0.00641	0.0509	0.0133	0.0686	0.0197
	Exact (box)	0.0169	0.00413	0.0361	0.0102	0.0635	0.0179
	Quadrature ($q = 2$)	0.0169	0.00415	0.036	0.0102	0.0636	0.0179
	Quadrature ($q = 3$)	0.0169	0.00414	0.036	0.0102	0.0635	0.0179

the quadrature-based evaluation, $q = 2$ and $q = 3$ denote using 2-point and 3-point per dimension, respectively. The two settings produce nearly identical results.

In 3D (Table 4.2), the radially symmetric approximation still reduces both the peak and RMS velocities, but less effectively than in 2D, whereas exact evaluation and quadrature-based evaluation remain fully effective. As in 2D, the quadrature-based evaluation produces nearly identical results for $q = 2$ and $q = 3$. We conclude that for box-shaped domains, 2-point quadrature per dimension suffices, so the cost of quadrature-based target evaluation can be kept low without sacrificing accuracy.

Finally, Figure 4.5 visualizes the effect in 2D at water level 0.75 under the slip boundary condition, showing frames 0, 30, 60, and 100 of the uncorrected baseline and of the position correction with quadrature-based targets ($q = 2$). In the baseline, parasitic motion builds up over time: the particle distribution grows increasingly uneven and the free

Table 4.2: v_{peak} and v_{rms} over 100 frames of the 3D hydrostatic test at resolution 32, for three water levels and two wall boundary conditions.

BC	Target evaluation	Water level					
		0.25		0.5		0.75	
		v_{peak}	v_{rms}	v_{peak}	v_{rms}	v_{peak}	v_{rms}
slip	None (baseline)	0.0471	0.0209	0.156	0.13	0.231	0.194
	Radial approximation	0.0639	0.0181	0.0908	0.0569	0.102	0.0918
	Exact (box)	0.0176	0.00431	0.0372	0.0111	0.0646	0.0197
	Quadrature ($q = 2$)	0.0176	0.00433	0.0373	0.0111	0.0643	0.0198
	Quadrature ($q = 3$)	0.0175	0.00431	0.0372	0.0111	0.0643	0.0197
sep	None (baseline)	0.0464	0.0213	0.156	0.131	0.233	0.195
	Radial approximation	0.064	0.0183	0.0908	0.0567	0.102	0.0915
	Exact (box)	0.0157	0.00393	0.037	0.0107	0.064	0.0195
	Quadrature ($q = 2$)	0.0159	0.00396	0.037	0.0107	0.0639	0.0194
	Quadrature ($q = 3$)	0.0157	0.00394	0.037	0.0107	0.0639	0.0194

surface becomes ragged. With the correction, the particle distribution remains uniform and the free surface stays flat throughout the simulation.

4.4.4 Volume Correction versus Position Correction

In addition to the position-fix results presented above, Table 4.3 compares the volume-fix against the position-fix of Section 4.2.4 on the hydrostatic test in 2D and 3D, at three water levels under slip boundary conditions. All runs use quadrature-based targets with $q = 2$. The volume fix uses a single regularization weight $\lambda = 0.01$ for all cases; the position fix uses its own regularization weight and parameter settings as specified in Section 4.4.3.

At this setting the two corrections perform comparably: both suppress the boiling to the comparable level across all water levels and both dimensions, with the position correction slightly ahead throughout. The distinction between the two methods appears

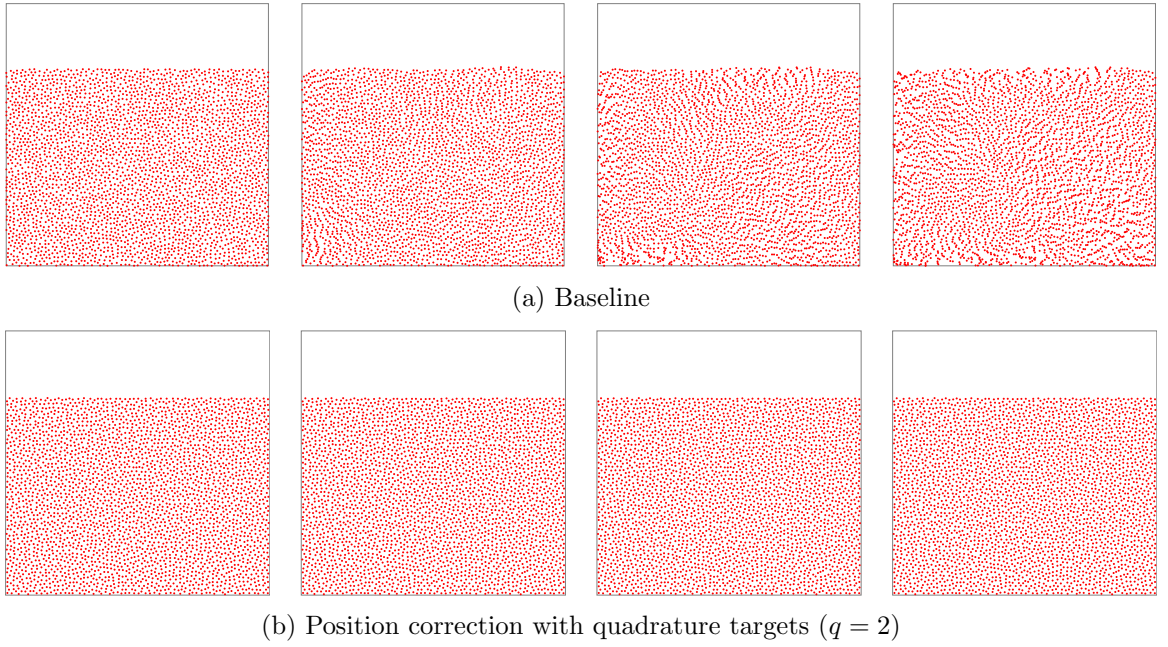


Figure 4.5: Hydrostatic test in 2D at water level 0.75 under the slip boundary condition, shown at frames 0, 30, 60, and 100 (left to right). (a) In the uncorrected baseline, visible flow patterns emerge and progressively disturb the particle distribution and the free surface. (b) With the position correction and quadrature-based target evaluation ($q = 2$), the particles stay at rest: the distribution remains uniform and the free surface stays flat.

when stronger correction is desired. Decreasing λ tightens the fit to the targets, but for the volume correction it also enlarges the deviations from the seed volumes, and eventually some particle volumes turn negative: at $\lambda = 10^{-3}$ with water level 0.75, negative volumes appear in both 2D and 3D. The accuracy of the volume correction is therefore capped by the positivity requirement. This is why the position correction serves as the main approach of this work.

Table 4.3: Volume correction versus position correction on the hydrostatic test (resolution 32, slip boundary conditions, three water levels). All cases use quadrature-based target evaluation with $q = 2$.

		Water level					
		0.25		0.5		0.75	
	Fix	v_{peak}	v_{rms}	v_{peak}	v_{rms}	v_{peak}	v_{rms}
2D	Volume	0.0187	0.00445	0.0382	0.011	0.0645	0.0213
	Position	0.018	0.00439	0.0363	0.0105	0.0639	0.0181
3D	Volume	0.0196	0.0048	0.0404	0.0133	0.065	0.0248
	Position	0.0176	0.00433	0.0373	0.0111	0.0643	0.0198

4.4.5 Node Volume Error Before and After Correction

The preceding sections established that correction effectively suppresses the boiling artifact; here we visualize the underlying volume change directly on the grid, which highlights the importance of the correction both in the interior and near the boundaries. Figure 4.6 examines the spatial distribution of the volume match on the 2D hydrostatic test (unit box, water level 0.75, slip boundary, resolution 32), using exact targets and position correction with $\hat{\lambda} = 10^{-6}$ and Newton tolerance 10^{-6} . Node volumes are normalized by the cell volume h^2 , and nodes with zero target are masked in gray throughout the figure.

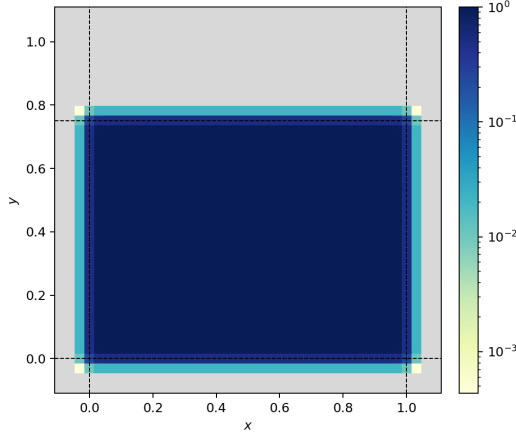
Figure 4.6, Panel (a) shows the targets themselves on a logarithmic scale: interior nodes carry one full cell, wall nodes one half, corner nodes one quarter, and nodes one cell outside the domain the small positive values implied by the B-spline support, visible only because of the logarithmic scale. Panel (b) shows the signed error after seeding and before correction,

$$e_i = \frac{1}{h^2} \left(\sum_p w_{ip}(\mathbf{X}^0) V_p - b_i \right),$$

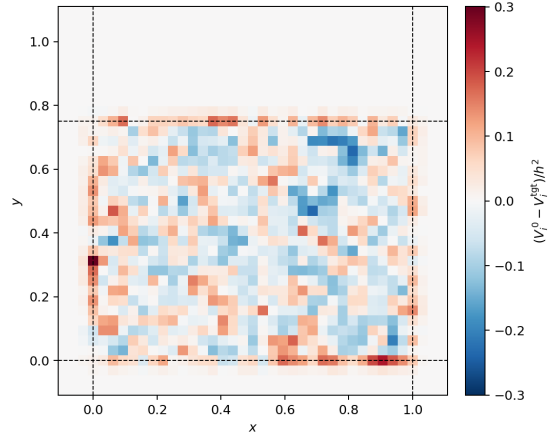
which exhibits the two error populations of Section 4.2.2: zero-mean sampling noise across the interior, and a systematic positive bias on the wall nodes, where near-boundary particles splat weight across the domain boundary. The maximum absolute error is 3.0×10^{-1} with RMS 5.6×10^{-2} .

Panel (c) shows the same signed error after the position correction. Both error populations are eliminated: the maximum absolute error drops to 4.3×10^{-4} and the RMS to 1.2×10^{-5} , a reduction of roughly three orders of magnitude; note that the color scale differs from panel (b) accordingly. Panel (d) plots the absolute error after correction on a logarithmic color scale to expose its full dynamic range. The largest remaining residual sits at a ghost node adjacent to the intersection of the free surface and the container wall, where the target volume is smallest but positive.

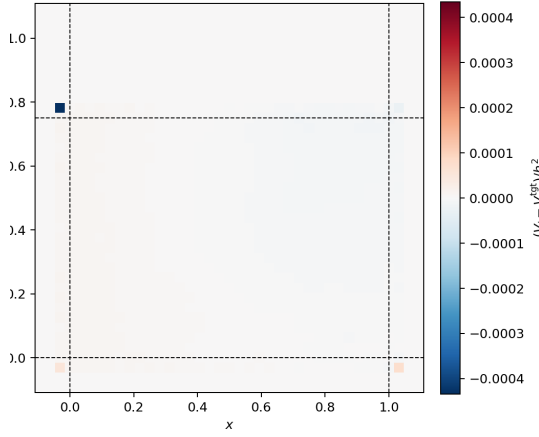
Figure 4.7 further shows how the correction reaches this state, plotting the residual energy $\frac{1}{2} \|\mathbf{r}(\mathbf{X}_k)\|^2$, without the penalty term, over the Newton iterations, for both dimensions and all three water levels under the same setup. The first three iterations do the bulk of the work, reducing the residual by five to six orders of magnitude. The 2D runs then converge below the tolerance within eight or nine iterations, while the 3D runs flatten near



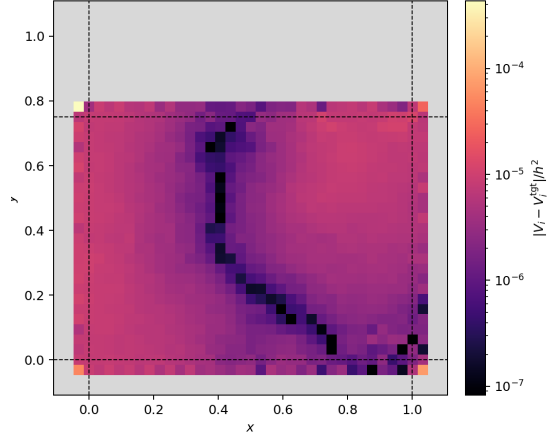
(a) Target node volumes V_i^{tgt}/h^2 .



(b) Signed error after seeding, before correction.



(c) Signed error after position correction.



(d) Absolute error after correction, $\log_{10}$ scale.

Figure 4.6: Node volumes for the 2D hydrostatic test (unit box, water level 0.75, resolution 32), normalized by h^2 ; nodes with zero target are masked in gray. (a) Exact tensor-product targets, logarithmic scale; dashed lines mark the container walls and the free surface. (b) Signed error after Poisson-disk seeding, before correction. (c) Signed error after position correction; note the color scale differs from (b). (d) Absolute error after correction, logarithmic scale.

10^{-5} and terminate at the iteration cap of ten without reaching the tolerance. This confirms the remark of Section 4.2.4: the tolerance is deliberately strict and need not be reached, since the residual level attained by ten iterations is already sufficient for the volume match to eliminate the artifact, as the preceding results show.

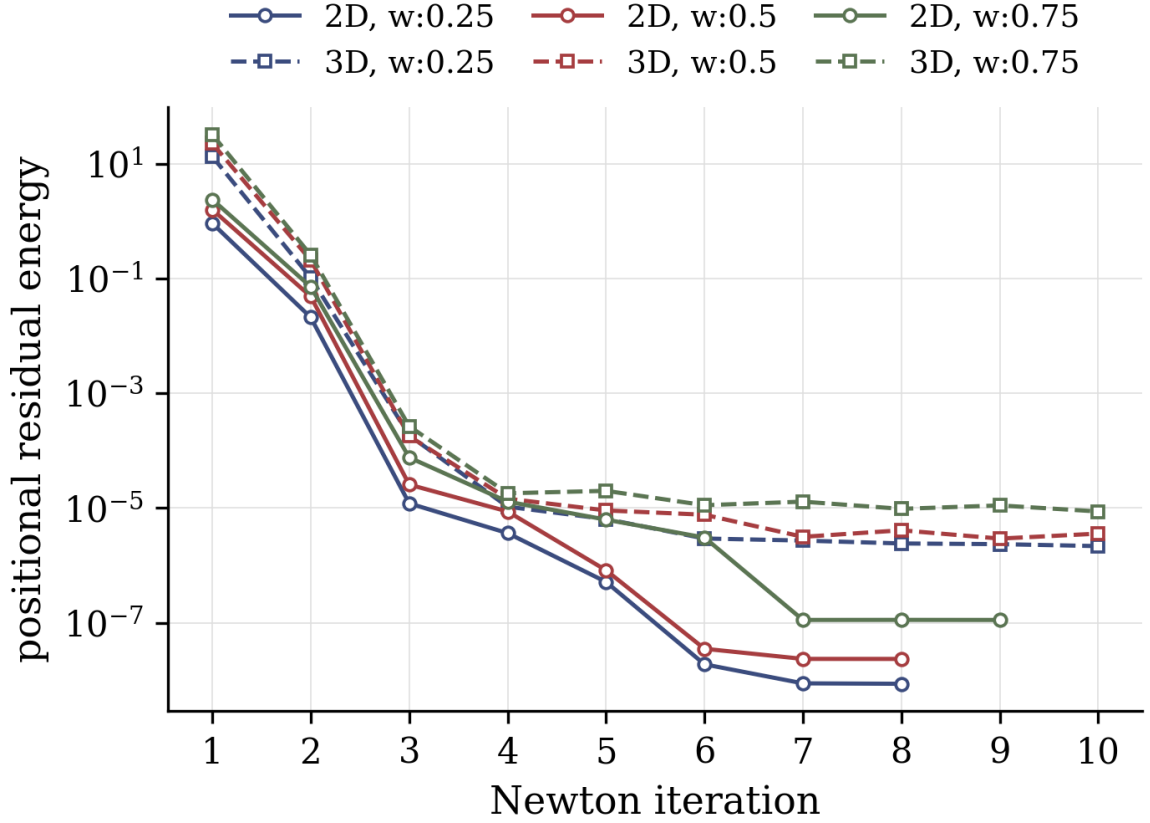


Figure 4.7: Residual energy $\frac{1}{2}\|\mathbf{r}(\mathbf{X}_k)\|^2$ of the position correction versus Newton iteration on the hydrostatic test (resolution 32, exact targets), for three water levels in 2D (solid) and 3D (dashed). The 2D runs converge below the tolerance within nine iterations; the 3D runs stop at the iteration cap of ten with residuals near 10^{-5} , which is nevertheless sufficient to eliminate the artifact

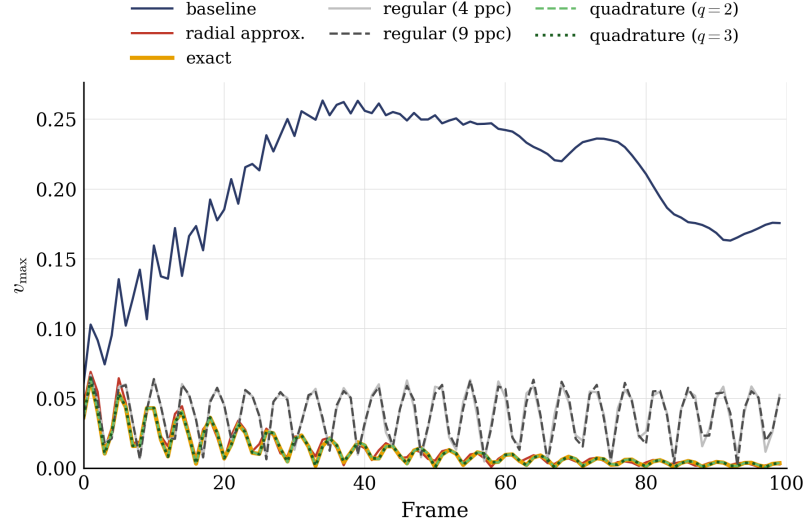
In summary, the position correction removes both the interior sampling noise and the boundary bias of the seeded state, the very mismatch that drives the boiling effect,

reducing the node volume error by roughly three orders of magnitude within ten Newton iterations.

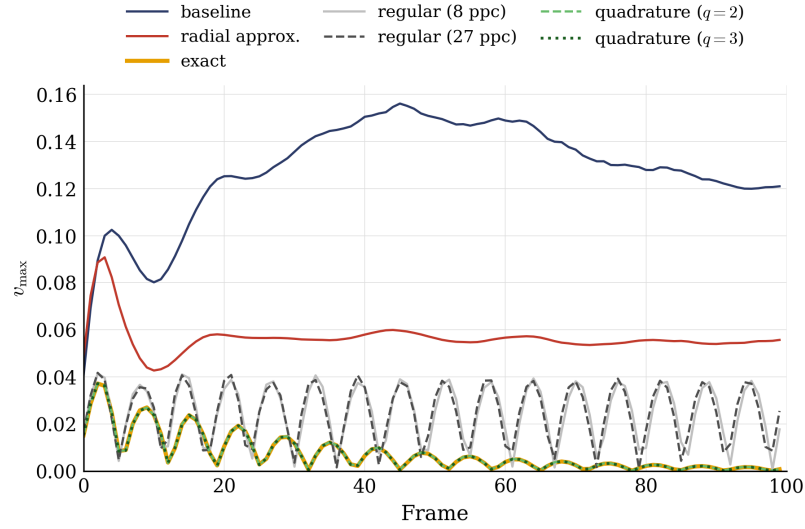
4.4.6 Peak Velocity Evolution and Comparison to Lattice Seeding

To continuously track how the boiling effect develops or diminishes over time, we plot the per-frame maximum particle speed, $v_{\max}(t_k)$, over 100 consecutive frames. Figure 4.8 presents the results for the hydrostatic test at resolution 32, with a water level of 0.75 in 2D and 0.5 in 3D. Seven configurations are shown: the uncorrected baseline; the three target evaluation methods of Section 4.3, each with the position correction (radial approximation, exact, and quadrature at $q = 2$ and $q = 3$); and regular lattice seeding as an ablation, in which particles are placed on a uniform lattice with uniform volumes and no correction. The lattice counts, 4 and 9 particles per cell in 2D (8 and 27 in 3D), are chosen so that the seeded lattices coincide with the quadrature lattices at $q = 2$ and $q = 3$, and the initial volume match therefore holds by construction without any solve.

The baseline climbs from the first frames and never settles, reaching 0.26 in 2D and 0.16 in 3D. The exact and quadrature curves are almost overlapping throughout: the maximum pointwise difference over all 100 frames is 6.9×10^{-4} ($q = 2$) and 2.8×10^{-4} ($q = 3$) in 2D, and 3.4×10^{-4} and 1.9×10^{-4} in 3D, and all of them decay toward zero. The radial approximation shows different behavior in 2D and 3D. In 2D, its curve remains close to the exact solution and decreases at a similar rate. In 3D, however, it deviates from the exact solution early and then stabilizes at approximately 0.06. This persistent nonzero value reflects the target approximation error near the edges and corners of the box.



(a) 2D, water level 0.75



(b) 3D, water level 0.5

Figure 4.8: Maximum velocity over time for the hydrostatic test on a 32^d grid, comparing target evaluation methods. The quadrature-based targets ($q = 2, 3$) are indistinguishable from the exact tensor-product targets, while the radial approximation and regular seeding retain residual motion.

The lattice runs also suppress the boiling, as expected from their built-in initial match, but their behavior differs in character from the corrected irregular samples. Instead of decaying, the lattice runs sustain a periodic oscillation around 0.04–0.06 that persists over the whole run and is not reduced by the finer lattice. Regular sampling is also known to exhibit undesired patterns in practical applications, which is why our method targets irregular samples in the first place. The corrected Poisson-disk configuration reaches a lower velocity that continues to decay over time, while preserving the blue-noise distribution of the particle sample. Figure 4.9 illustrates this contrast directly: the lattice sample is perfectly regular, while the Poisson-disk sample before and after correction is nearly indistinguishable to the eye. The correction achieves the volume match through small positional adjustments that leave the irregular, evenly spaced character of the sample intact.

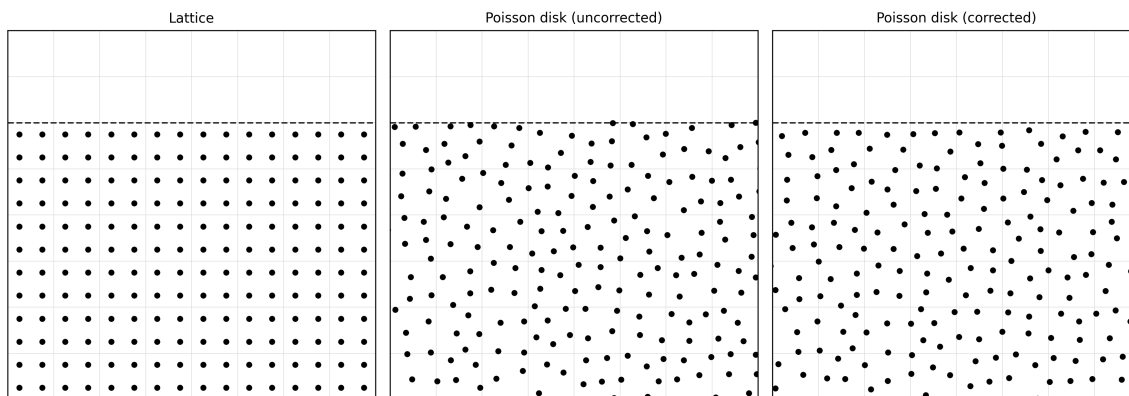


Figure 4.9: Close-up of the initial particle distribution for the 2D hydrostatic test (unit box, water level 0.75, resolution 32). Each panel shows the same 8×8 -cell region of the domain, spanning the free surface: the dashed line marks the initial water level, with water below and empty cells above, and light gray lines indicate the grid cells. Left: regular lattice seeding with four particles per cell. Center: Poisson-disk sampling before correction. Right: the same Poisson-disk sample after position correction with quadrature-based targets ($q = 2$). The correction moves particles only slightly and preserves the irregular, evenly spaced character of the sample.

4.4.7 Dam Break

The hydrostatic test isolates the initialization artifact, but its exact solution is trivial: nothing should move. To confirm that the correction does not disturb a genuinely dynamic simulation, we run a dam break test. A column of water is released at one side of a box and collapses under gravity; the front travels along the floor, strikes the opposite wall, and the fluid sloshes until it comes to rest (Figure 4.10). Unlike the hydrostatic test, the initial geometry is transient: the configuration the correction acts on is destroyed within the first frames of the simulation. The purpose of the experiment is therefore not to measure a reduction of the artifact, but to verify that the correction introduces no spurious behavior of its own.

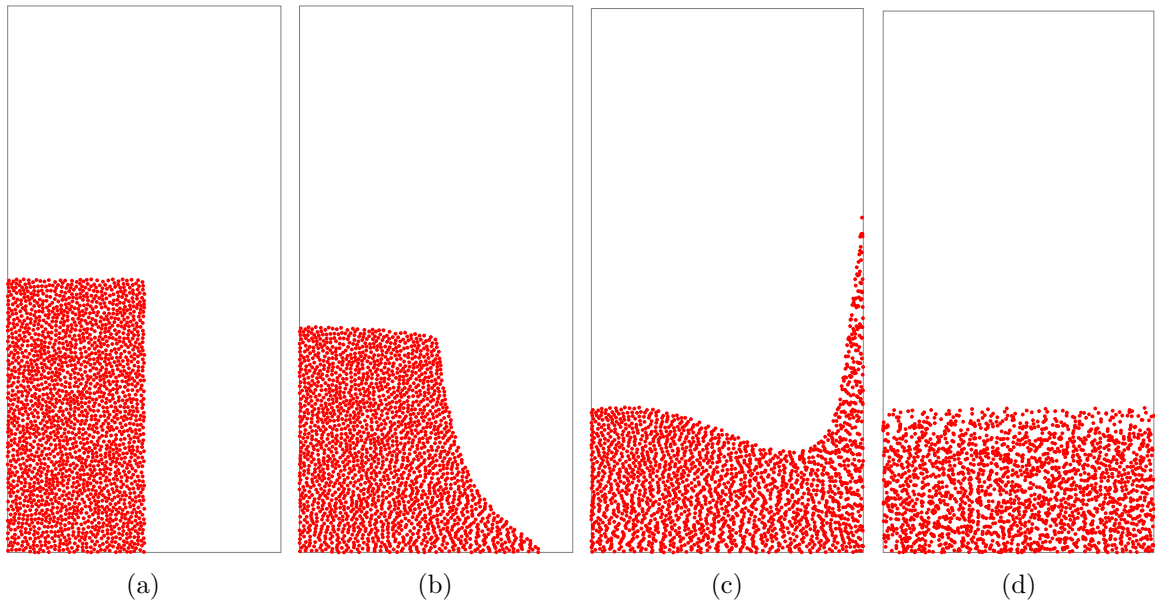


Figure 4.10: Dam break test: (a) initial configuration, (b) collapse, (c) impact against the right wall, (d) near rest. The water column of width 0.5 and height 1 is released in a box of height 2; frames from the uncorrected run.

The domain is a box of width 1 and height 2, discretized at resolution 32 (32×64 cells). The water column occupies the left half of the floor, with width 0.5 and height 1. Gravity, the pressure model, the seeding, and the correction parameters are those of Section 4.4.1 and 4.4.2, and the corrected run uses the position fix with quadrature-based targets ($q = 2$). The side and bottom walls use the slip boundary condition. The top wall uses the separating condition instead: under a slip ceiling, splashed particles that reach the top remain attached to it and slide along it, which contaminates the velocity statistics without affecting the flow.

Figure 4.11 plots the RMS and maximum particle velocity over the first 1000 frames, for the baseline and the corrected run. The two curves coincide through the collapse, the impact against the wall, and the subsequent sloshing decay. The difference in the initial particle positions produces no visible difference in the velocity evolution at any stage. We conclude that the correction is safe to apply as a default. It removes the artifact where the artifact exists, and leaves the dynamics unchanged where it does not.

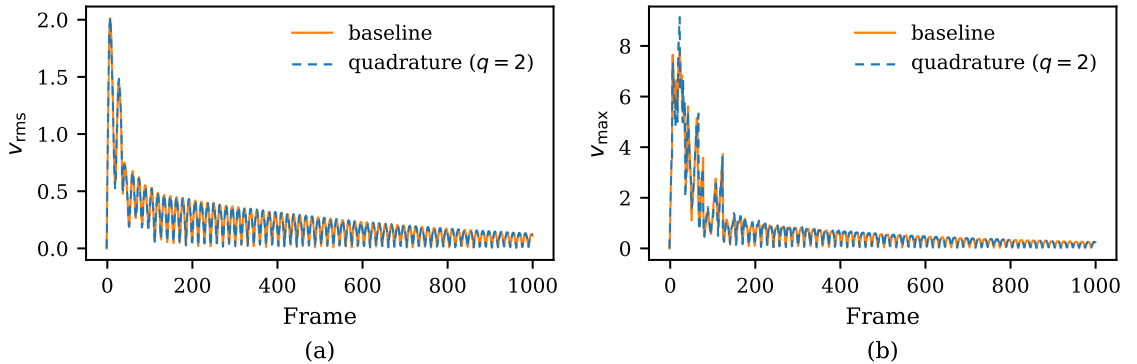


Figure 4.11: Dam break test at resolution (32): (a) RMS and (b) maximum particle velocity over the first 1000 frames. The baseline uses the original particle initialization, while the corrected run uses the position fix with quadrature-based target evaluation ($q = 2$). The two runs show almost the same velocity behavior during collapse, impact, and later sloshing, indicating that the correction does not change the overall dynamics.

4.4.8 Two-Density Stratification

The hydrostatic test above uses particles of uniform mass, leaving open whether the fix extends to varying particle masses. We address this with a two-density stratification test. The test also probes a consequence of the artifact that the previous experiments cannot show: spurious motion acting on a density interface progressively degrades the structure of the solution.

The domain is a box of width 2 and height 4, discretized at resolution 32 (32×64 cells), with slip boundary conditions on all four walls. Two fluids fill the box in a stably stratified arrangement: a dense fluid with $\rho_1 = 2$ occupies the bottom half ($y < 2$) and a light fluid with $\rho_0 = 1$ occupies the top half, separated by a sharp horizontal interface. Particles are seeded over the full domain by Poisson-disk sampling as in Section 4.4.1, and particles below the interface are assigned the dense mass. Gravity, the pressure model, and the correction parameters are those of Sections 4.4.1 and 4.4.2, and the corrected run uses the position fix with quadrature-based targets ($q = 2$). Each run is advanced for 300 frames.

Because the stratification is stable, the exact solution is static: the pressure gradient balances gravity in each layer, and the interface remains flat. Any sustained motion is therefore artifact, as in the hydrostatic test, but now in the presence of a discontinuous per-particle mass distribution.

Figure 4.12 plots the RMS and maximum particle velocity for both runs. The curves coincide over the first frames: the particles are seeded at zero pressure, so the fluid compresses slightly under gravity and excites a global oscillation that is independent of the initialization quality. In the corrected run this oscillation is the only motion present, and it

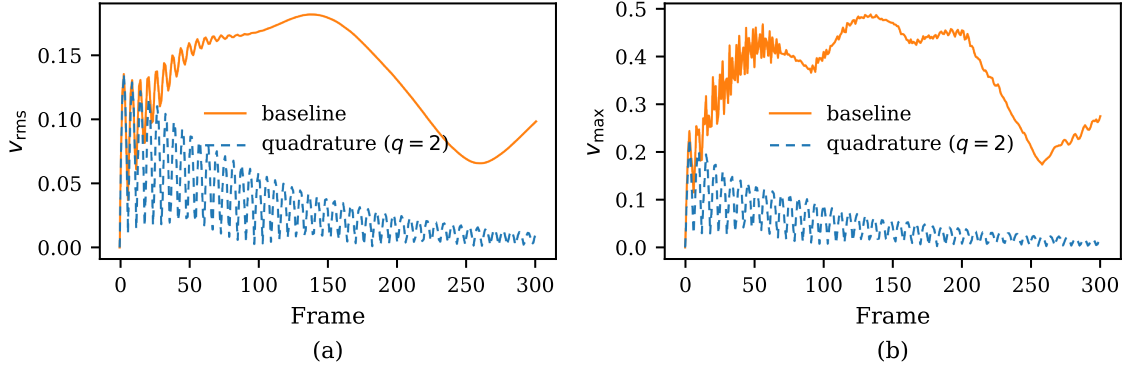
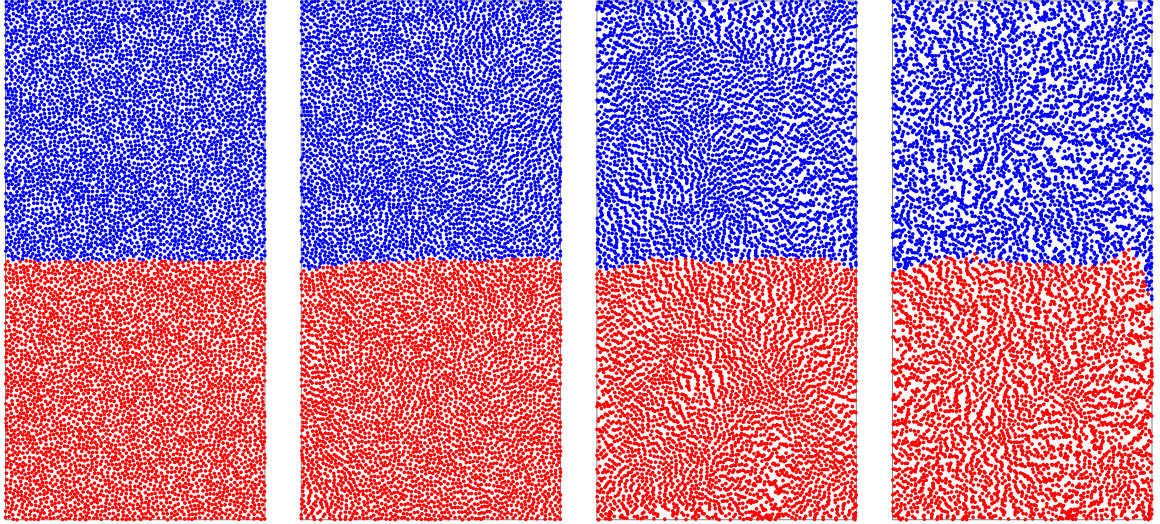


Figure 4.12: Sharp-interface two-density test: (a) root-mean-square and (b) maximum particle velocity over 300 frames, comparing baseline seeding with position correction using quadrature targets ($q = 2$). The exact solution is static. In the corrected run the initial settling oscillation decays steadily toward zero. In the baseline run the velocity remains large throughout: it does not decay monotonically but wanders, rising again after frame 250, as spurious motion continues to agitate the interface.

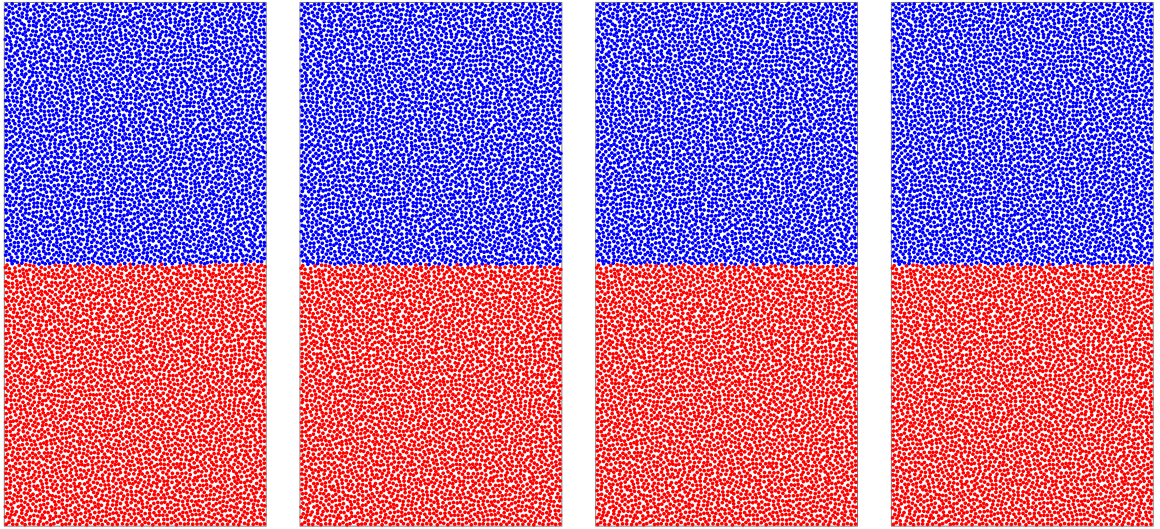
decays steadily toward zero. In the baseline run the velocity remains large throughout and does not decay monotonically, rising again after frame 250; the spurious motion is sustained rather than transient.

Figure 4.13 shows the particle distributions at frames 0, 30, 100, and 300. In the baseline run the interface roughens progressively: particles of each fluid are pushed across the interface, and the initially uniform particle distribution within each layer is visibly disturbed. In the corrected run the interface remains flat and the particle distribution is nearly unchanged over all 300 frames. The degradation in the baseline is driven entirely by the initialization error. No physical mechanism in the simulation acts to disturb a stably stratified configuration at rest.

We draw two conclusions. First, the correction applies effectively to particles of non-uniform mass. Second, the initialization error is not confined to the start of the



(a) Baseline



(b) Position correction with quadrature targets ($q = 2$)

Figure 4.13: Particle distributions at frames 0, 30, 100, and 300 (left to right) for the sharp-interface two-density test; the light fluid is blue and the dense fluid is red. In the baseline run (a) the interface roughens and the particle arrangement in both layers is visibly disturbed. In the corrected run (b) the interface remains flat and the particle distribution is nearly unchanged over all 300 frames.

simulation: the baseline run shows that its effect accumulates over hundreds of frames, while the corrected run remains close to the exact solution throughout.

4.5 Conclusion

This chapter addressed a persistent artifact in MPM fluid simulation: spurious velocities, or “boiling,” arising from inconsistent particle initialization. We showed that the standard seeding procedure produces a rasterized volume field that disagrees, node by node, with the volume each node represents under the B-spline transfer, and that this disagreement alone is sufficient to inject energy into a fluid that should remain at rest. Our contribution is an initialization stage that removes the inconsistency before the first time step by enforcing a volume matching condition at every node. We developed two solvers for this condition: a regularized linear correction of the particle volumes, solved matrix-free by conjugate gradient, and a Gauss–Newton correction of the particle positions with Wolfe line search, likewise matrix-free. The volume correction is capped by a positivity requirement, since decreasing the regularization weight eventually drives particle volumes negative, and the position correction, which cannot produce an invalid particle, therefore serves as the main approach.

In deriving the targets, we identified a consequential error in the radially symmetric closed-form approximation: built on a single signed distance, it cannot represent two intersecting walls, and it overestimates boundary-node volumes at edges and corners in 3D by factors of up to two and four. This overestimate is itself a source of residual boiling. Exact tensor-product node volumes eliminate the error on box domains, and a quadrature

evaluation at modest order ($q = 2\text{--}3$) reproduces the exact targets on the box using the existing gather–scatter infrastructure. The quadrature construction itself is not tied to the box and can be extended to general level-set geometries. We further nondimensionalized the solver parameters so that a single calibration, performed once on a coarse 2D run, transfers across resolutions and dimensions.

The method was validated through three main experiments of increasing dynamic complexity. In two- and three-dimensional hydrostatic water-column tests, the correction reduces nodal volume errors by approximately three orders of magnitude and brings the velocities close to equilibrium, whereas the baseline method exhibits persistent boiling. On a stratified two-density configuration, the correction extends to non-uniform particle masses and preserves the interface that the baseline progressively roughens. On a dam break, the corrected and uncorrected runs coincide, confirming that the correction introduces no spurious behavior of its own.

4.5.1 Limitations and Future Work

The experiments in this chapter are confined to axis-aligned box domains, and two directions remain open.

The first is irregular geometry, such as a spherical container. The quadrature targets already handle such shapes by construction, and the initialization solve itself carries over unchanged. The obstacle is the boundary handling at run time. Containers that coincide with the domain walls are enforced through the domain boundary condition, which processes boundary nodes in pairs: momentum removed from a node outside the wall is redistributed to its partner inside, so the grid retains the momentum that the volume-

matching targets account for. Containers that do not coincide with the domain walls must instead be enforced as collision objects, whose treatment projects each nodal velocity against the object independently, with no paired redistribution; the momentum removed at the boundary is simply discarded. This unpaired projection introduces an inconsistency at the boundary that persists regardless of the initial state, and our preliminary experiments indicate that the initialization fix alone does not remove the resulting artifact. Extending the validation to irregular containers therefore goes together with a collision treatment that conserves the momentum it currently discards, which we regard as the natural next step.

The second is the analytical evaluation of the targets for complex objects. The quadrature method is general but numerical; the radial approximation shows that a closed-form evaluation can be both cheap and accurate where its geometric assumption holds, and its failure at edges and corners is a failure of the single-distance form, not of the analytical approach as such. Closed-form constructions that account for more of the local boundary geometry, such as multiple distances or curvature information, would combine the convenience of the radial formula with the accuracy of the exact targets, and we regard this as a promising direction.

Chapter 5

Conclusion

In this dissertation, we have presented methods for two problems that arise where grid data meets a bounded domain: the interpolation of divergence-free vector fields from MAC grid data over irregular domains, and the initialization of consistent particle states for the Material Point Method (MPM).

For interpolation, we constructed divergence-free schemes over triangles and squares in two dimensions and over tetrahedra and cubes in three dimensions, completing the incomplete divergence-free polynomial basis with rational functions. In three dimensions, cut cells are divided by a specially constructed marching-cubes table that produces non-inverted tetrahedra matching continuously between grid cells. These schemes fill the gap region between an irregular boundary and the interior where spline-based schemes apply, yielding an interpolant that is analytically divergence-free, C^0 continuous everywhere, and purely local.

For initialization, we formulated the inconsistency behind the boiling effect as the residual between the particle-rasterized nodal volumes and the target nodal volumes of the

domain discretization. The residual can be minimized through either set of unknowns: a linear volume correction solved as a regularized least-squares system, whose accuracy is capped by the requirement that particle volumes remain positive, or a nonlinear position redistribution solved with a Gauss–Newton method, which cannot produce invalid volumes and serves as our main approach. Both stages run once, before the first time step, and the corrected initial states substantially reduce the boiling effect relative to the uncorrected initialization. Applying the method to square and cube domains with water showed that the accuracy of the target volumes is decisive where walls meet: with exact nodal volumes, computed in closed form for these domains, the correction successfully stops the boiling effect in both two and three dimensions, whereas a radially symmetric approximation of the nodal volume is effective only in two dimensions, leaving spurious motion along the edges of the domain box in three dimensions. A quadrature evaluation of the targets reproduces the exact values at modest order and extends the construction beyond the box geometry.

In both problems, a property that holds automatically on a regular interior fails at the boundary, and the remedy is to impose the missing structure explicitly — exactly, by construction, or approximately, by minimization. The two works differ in how far this program is carried: the interpolation handles general irregular boundaries, while the initialization is demonstrated on grid-aligned domains, with its extension to irregular geometry outlined as future work.

Several directions remain. The rational basis functions are not differentiable at element corners in two dimensions and along element edges in three dimensions, and the discretization does not optimize element conditioning; a conditioning-aware discretization of the gap region would improve the reconstruction. The interpolation scheme also de-

depends on a spline-based interpolant in the interior, an assumption that could possibly be relaxed later. For initialization, the exact nodal volumes that stop the boiling effect admit closed forms only for simple geometries, and while the quadrature evaluation extends the targets beyond the box, an analytical construction that is both efficient to evaluate and accurate where boundaries meet at edges and corners, precisely where the radially symmetric approximation fails, remains open. Carrying the validation to irregular containers further requires a consistent boundary treatment for containers that do not coincide with the domain walls. Finally, there is the mismatch between the stencils through which gravity and stress act: gravity acts through the weighted nodal mass, whereas stress acts through the weight gradients, and addressing this gap directly may further suppress the remaining artifacts.

Bibliography

- [1] Duarte MS Albuquerque, Diogo MC Martins, and José CF Pereira. An immersed boundary method with divergence-free interpolation for unstructured polyhedral grids. In *6th ECCOMAS European Conference on Computational Mechanics: Solids, Structures and Coupled Problems, ECCM 2018 and 7th ECCOMAS European Conference on Computational Fluid Dynamics, ECFD 2018*, pages 3429–3439. International Centre for Numerical Methods in Engineering, CIMNE, 2020.
- [2] Osama Alkhateeb and Igor Tsukerman. A boundary difference method for electromagnetic scattering problems with perfect conductors and corners. *IEEE Transactions on Antennas and Propagation*, 61(10):5117–5126, 2013. doi: 10.1109/TAP.2013.2275152.
- [3] Luca Amodei and Mohamed-Najib Benbourhim. A vector spline approximation. *Journal of approximation theory*, 67(1):51–79, 1991.
- [4] Dinshaw S Balsara. Divergence-free adaptive mesh refinement for magnetohydrodynamics. *Journal of Computational Physics*, 174(2):614–648, 2001.
- [5] Dinshaw S Balsara. Second-order-accurate schemes for magnetohydrodynamics with divergence-free reconstruction. *The Astrophysical Journal Supplement Series*, 151(1):149, 2004.
- [6] Dinshaw S Balsara. Divergence-free reconstruction of magnetic fields and weno schemes for magnetohydrodynamics. *Journal of Computational Physics*, 228(14):5040–5056, 2009.
- [7] Dinshaw S Balsara and Michael Dumbser. Divergence-free mhd on unstructured meshes using high order finite volume schemes based on multidimensional riemann solvers. *Journal of Computational Physics*, 299:687–715, 2015.
- [8] Dinshaw S Balsara and Jongsoo Kim. A comparison between divergence-cleaning and staggered-mesh formulations for numerical magnetohydrodynamics. *The Astrophysical Journal*, 602(2):1079, 2004.
- [9] Dinshaw S Balsara and Daniel S Spicer. A staggered mesh algorithm using high order godunov fluxes to ensure solenoidal magnetic fields in magnetohydrodynamic simulations. *Journal of Computational Physics*, 149(2):270–292, 1999.

- [10] Yuanxun Bao, Aleksandar Donev, Boyce E Griffith, David M McQueen, and Charles S Peskin. An immersed boundary method with divergence-free velocity interpolation and force spreading. *Journal of computational physics*, 347:183–206, 2017.
- [11] SG. Bardenhagen and EM. Kober. The generalized interpolation material point method. *Comp Mod in Eng and Sci*, 5(6):477–496, 2004.
- [12] Gabriel R Barrenechea, Erik Burman, Ernesto Cáceres, and Johnny Guzmán. Continuous interior penalty stabilization for divergence-free finite element methods. *IMA Journal of Numerical Analysis*, 44(2):980–1002, 2024.
- [13] Aaron S Baumgarten and Ken Kamrin. Analysis and mitigation of spatial integration errors for the material point method. *International Journal for Numerical Methods in Engineering*, 124(11):2449–2497, 2023.
- [14] Ted Belytschko, Wing Kam Liu, Brian Moran, and Khalil Elkhodary. *Nonlinear finite elements for continua and structures*. John wiley & sons, 2014.
- [15] Mohammed-Najib Benbourhim and Abderrahman Bouhamidi. Meshless pseudo-polyharmonic divergence-free and curl-free vector fields approximation. *SIAM journal on mathematical analysis*, 42(3):1218–1245, 2010.
- [16] Kai Bittner and Karsten Urban. On interpolatory divergence-free wavelets. *Mathematics of computation*, 76(258):903–929, 2007.
- [17] Parikshit Boregowda and Gui-Rong Liu. A boundary integral based particle initialization algorithm for smooth particle hydrodynamics. *Computer Physics Communications*, 310:109531, 2025. ISSN 0010-4655. doi: <https://doi.org/10.1016/j.cpc.2025.109531>. URL <https://www.sciencedirect.com/science/article/pii/S0010465525000347>.
- [18] Walter Boscheri, Giuseppe Roberto Pisaturo, and Maurizio Righetti. High-order divergence-free velocity reconstruction for free surface flows on unstructured voronoi meshes. *International Journal for Numerical Methods in Fluids*, 90(6):296–321, 2019.
- [19] R. Bridson. Fast poisson disk sampling in arbitrary dimensions. In *SIGGRAPH sketches*, page 22, 2007.
- [20] Jumyung Chang, Ruben Partono, Vinicius C. Azevedo, and Christopher Batty. Curl-flow: Boundary-respecting pointwise incompressible velocity interpolation for grid-based fluids. *ACM Transactions on Graphics*, 41(6):1–21, November 2022. ISSN 1557-7368. doi: 10.1145/3550454.3555498. URL <http://dx.doi.org/10.1145/3550454.3555498>.
- [21] Long Chen and Feng Wang. A divergence free weak virtual element method for the stokes problem on polytopal meshes. *Journal of Scientific Computing*, 78:864–886, 2019.
- [22] Alexandre Joel Chorin. Numerical solution of the navier-stokes equations. *Mathematics of computation*, 22(104):745–762, 1968.

- [23] Andrea Colagrossi, B Bouscasse, Matteo Antuono, and Salvatore Marrone. Particle packing algorithm for sph schemes. *Computer Physics Communications*, 183(8):1641–1653, 2012.
- [24] G. Daviet and F. Bertails-Descoubes. A semi-implicit material point method for the continuum simulation of granular materials. *ACM Trans Graph*, 35(4), jul 2016.
- [25] Alban de Vaucorbeil, Vinh Phu Nguyen, Sina Sinaie, and Jian Ying Wu. Material point method after 25 years: theory, implementation and applications. *Submitted to Advances in Applied Mechanics*, page 1, 2019.
- [26] Erwan Deriaz and Valérie Perrier. Towards a divergence-free wavelet method for the simulation of 2d/3d turbulent flows. *J. Turbul*, 7(3):1–37, 2006.
- [27] Fabrice Dodu and Christophe Rabut. Irrotational or divergence-free interpolation. *Numerische Mathematik*, 98(3):477–498, 2004.
- [28] Kathryn P Drake, Edward J Fuselier, and Grady B Wright. A partition of unity method for divergence-free or curl-free radial basis function approximation. *SIAM Journal on Scientific Computing*, 43(3):A1950–A1974, 2021.
- [29] Michael Dumbser, Dinshaw S Balsara, Maurizio Tavelli, and Francesco Fambri. A divergence-free semi-implicit finite volume scheme for ideal, viscous, and resistive magnetohydrodynamics. *International Journal for Numerical Methods in Fluids*, 89(1-2): 16–42, 2019.
- [30] Patricio Farrell, Kathryn Gillow, and Holger Wendland. Multilevel interpolation of divergence-free vector fields. *IMA Journal of Numerical Analysis*, 37(1):332–353, 05 2016. ISSN 0272-4979. doi: 10.1093/imanum/drw006. URL <https://doi.org/10.1093/imanum/drw006>.
- [31] John M Finn and Luis Chacón. Volume preserving integrators for solenoidal fields on a grid. *Physics of Plasmas*, 12(5):054503, 2005.
- [32] Thomas Frachon, Peter Hansbo, Erik Nilsson, and Sara Zahedi. A divergence preserving cut finite element method for darcy flow. *SIAM Journal on Scientific Computing*, 46(3):A1793–A1820, 2024.
- [33] Derk Frerichs and Christian Merdon. Divergence-preserving reconstructions on polygons and a really pressure-robust virtual element method for the stokes problem, 2020. URL <https://arxiv.org/abs/2002.01830>.
- [34] Johnny Guzmán and Michael Neilan. Conforming and divergence-free stokes elements in three dimensions. *IMA Journal of Numerical Analysis*, 34(4):1489–1508, 2014.
- [35] Johnny Guzmán and Michael Neilan. Conforming and divergence-free stokes elements on general triangular meshes. *Mathematics of Computation*, 83(285):15–36, 2014.
- [36] Johnny Guzmán and Michael Neilan. Symmetric and conforming mixed finite elements for plane elasticity using rational bubble functions. *Numerische Mathematik*, 126:153–171, 2014.

- [37] DC Handscomb. Spline representation of incompressible flow. *IMA journal of numerical analysis*, 4(4):491–502, 1984.
- [38] F. Harlow and E. Welch. Numerical calculation of time dependent viscous flow of fluid with a free surface. *Phys Fluid*, 8(12):2182–2189, 1965.
- [39] Ralf Hiptmair, Lingxiao Li, Shipeng Mao, and Weiyang Zheng. A fully divergence-free finite element method for magnetohydrodynamic equations. *Mathematical Models and Methods in Applied Sciences*, 28(04):659–695, 2018.
- [40] Yuanming Hu, Yu Fang, Ziheng Ge, Ziyin Qu, Yixin Zhu, Andre Pradhana, and Chenfanfu Jiang. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Transactions on Graphics (TOG)*, 37(4):150, 2018.
- [41] Patrick Jenny, Stephen B Pope, Metin Muradoglu, and David A Caughey. A hybrid algorithm for the joint pdf equation of turbulent reactive flows. *Journal of Computational Physics*, 166(2):218–252, 2001.
- [42] C. Jiang, C. Schroeder, A. Selle, J. Teran, and A. Stomakhin. The affine particle-in-cell method. *ACM Trans Graph*, 34(4):51:1–51:10, 2015.
- [43] Yingchun Jiang. Anisotropic curl-free wavelets with boundary conditions. *Journal of Inequalities and Applications*, 2012:1–10, 2012.
- [44] Jing Jin and Craig Schroeder. Divergence-free interpolation of mac grid data over irregular domains. *Journal of Computational Physics*, 560:114924, 2026. ISSN 0021-9991. doi: <https://doi.org/10.1016/j.jcp.2026.114924>. URL <https://www.sciencedirect.com/science/article/pii/S0021999126002743>.
- [45] Volker John, Alexander Linke, Christian Merdon, Michael Neilan, and Leo G Rebholz. On the divergence constraint in mixed finite element methods for incompressible flows. *SIAM review*, 59(3):492–544, 2017.
- [46] Kamau Kingora and Hamid Sadat-Hosseini. A novel interpolation-free sharp-interface immersed boundary method. *Journal of computational physics*, 453:110933, 2022.
- [47] G. Klár, T. Gast, A. Pradhana, C. Fu, C. Schroeder, C. Jiang, and T. Teran. Drucker-prager elastoplasticity for sand animation. *ACM Transactions on Graphics (SIGGRAPH 2016)*., 2016.
- [48] Philip L Lederer and Sander Rhebergen. A pressure-robust embedded discontinuous galerkin method for the stokes problem by reconstruction operators. *SIAM Journal on Numerical Analysis*, 58(5):2915–2933, 2020.
- [49] Christoph Lehrenfeld and Joachim Schöberl. High order exactly divergence-free hybrid discontinuous galerkin methods for unsteady incompressible flows. *Computer Methods in Applied Mechanics and Engineering*, 307:339–361, 2016.

- [50] Xu Li and Hongxing Rui. A low-order divergence-free h (div)-conforming finite element method for stokes flows. *IMA Journal of Numerical Analysis*, 42(4):3711–3734, 2022.
- [51] S.J. Lind, R. Xu, P.K. Stansby, and B.D. Rogers. Incompressible smoothed particle hydrodynamics for free-surface flows: A generalised diffusion-based algorithm for stability and validations for impulsive flows and propagating waves. *Journal of Computational Physics*, 231(4):1499–1523, 2012. ISSN 0021-9991. doi: <https://doi.org/10.1016/j.jcp.2011.10.027>. URL <https://www.sciencedirect.com/science/article/pii/S0021999111006279>.
- [52] Sergey Litvinov, XY Hu, and Nikolaus A Adams. Towards consistence and convergence of conservative sph approximations. *Journal of Computational Physics*, 301:394–401, 2015.
- [53] Haoran Liu, Michael Neilan, and Maxim Olshanskii. A cutfem divergence-free discretization for the stokes problem. *ESAIM: Mathematical Modelling and Numerical Analysis*, 57(1):143–165, 2023.
- [54] Svenja Lowitzsch. Matrix-valued radial basis functions: stability estimates and applications. *Advances in Computational Mathematics*, 23(3):299–315, 2005.
- [55] F Mackay, R Marchand, and K Kabin. Divergence-free magnetic field interpolation and charged particle trajectory integration. *Journal of Geophysical Research: Space Physics*, 111(A6), 2006.
- [56] Salvatore Marrone, MAGD Antuono, A Colagrossi, G Colicchio, D Le Touzé, and Giorgio Graziani. δ -sph model for simulating violent impact flows. *Computer Methods in Applied Mechanics and Engineering*, 200(13-16):1526–1542, 2011.
- [57] Colin P McNally. Divergence-free interpolation of vector fields from point values—exact $\nabla \cdot \mathbf{B} = 0$ in numerical simulations. *Monthly Notices of the Royal Astronomical Society: Letters*, 413(1):L76–L80, 2011.
- [58] DW Meyer and P Jenny. Conservative velocity interpolation for pdf methods. In *PAMM: Proceedings in Applied Mathematics and Mechanics*, volume 4, pages 466–467. Wiley Online Library, 2004.
- [59] Rajat Mittal and Gianluca Iaccarino. Immersed boundary methods. *Annual Review of Fluid Mechanics*, 37(Volume 37, 2005):239–261, 2005. ISSN 1545-4479. doi: <https://doi.org/10.1146/annurev.fluid.37.061903.175743>. URL <https://www.annualreviews.org/content/journals/10.1146/annurev.fluid.37.061903.175743>.
- [60] Joe J Monaghan. Smoothed particle hydrodynamics. In: *Annual review of astronomy and astrophysics. Vol. 30 (A93-25826 09-90), p. 543-574.*, 30:543–574, 1992.
- [61] Francis J Narcowich and Joseph D Ward. Generalized hermite interpolation via matrix-valued conditionally positive definite functions. *Mathematics of Computation*, 63(208): 661–687, 1994.

- [62] Pawan Negi and Prabhu Ramachandran. Algorithms for uniform particle initialization in domains with complex boundaries. *Computer Physics Communications*, 265:108008, 2021. ISSN 0010-4655. doi: <https://doi.org/10.1016/j.cpc.2021.108008>. URL <https://www.sciencedirect.com/science/article/pii/S001046552100120X>.
- [63] Yen Ting Ng, Chohong Min, and Frédéric Gibou. An efficient fluid–solid coupling algorithm for single-phase flows. *Journal of Computational Physics*, 228(23):8807–8829, 2009.
- [64] Charles S. Peskin. The immersed boundary method. *Acta Numerica*, 11:479–517, 2002. doi: 10.1017/S0962492902000077.
- [65] Ramsharan Rangarajan, Adrián Lew, and Gustavo C. Buscaglia. A discontinuous-galerkin-based immersed boundary method with non-homogeneous boundary conditions and its application to elasticity. *Computer Methods in Applied Mechanics and Engineering*, 198(17):1513–1534, 2009. ISSN 0045-7825. doi: <https://doi.org/10.1016/j.cma.2009.01.018>. URL <https://www.sciencedirect.com/science/article/pii/S0045782509000413>.
- [66] Bharath Ravu, Murray Rudman, Guy Metcalfe, Daniel R Lester, and Devang V Khakhar. Creating analytically divergence-free velocity fields from grid-based data. *Journal of Computational Physics*, 323:75–94, 2016.
- [67] Ritoban Roy-Chowdhury, Tamar Shinar, and Craig Schroeder. Higher order divergence-free and curl-free interpolation on mac grids. *Journal of Computational Physics*, 503:112831, 2024.
- [68] A Sadeghirad, Rebecca M Brannon, and J Burghardt. A convected particle domain interpolation technique to extend applicability of the material point method for problems involving massive deformations. *International Journal for numerical methods in Engineering*, 86(12):1435–1456, 2011.
- [69] Dominik Schillinger, Isaac Harari, Ming-Chen Hsu, David Kamensky, Stein K.F. Stoter, Yue Yu, and Ying Zhao. The non-symmetric nitsche method for the parameter-free imposition of weak boundary and coupling conditions in immersed finite elements. *Computer Methods in Applied Mechanics and Engineering*, 309:625–652, 2016. ISSN 0045-7825. doi: <https://doi.org/10.1016/j.cma.2016.06.026>. URL <https://www.sciencedirect.com/science/article/pii/S0045782516306338>.
- [70] Craig Schroeder, Ritoban Roy Chowdhury, and Tamar Shinar. Local divergence-free polynomial interpolation on mac grids. *Journal of Computational Physics*, 468:111500, 2022. ISSN 0021-9991. doi: <https://doi.org/10.1016/j.jcp.2022.111500>. URL <https://www.sciencedirect.com/science/article/pii/S0021999122005629>.
- [71] Seungwon Shin and Damir Juric. A hybrid interface method for three-dimensional multiphase flows based on front tracking and level set techniques. *International Journal for Numerical Methods in Fluids*, 60(7):753–778, 2009.

- [72] Zachary J Silberman, Thomas R Adams, Joshua A Faber, Zachariah B Etienne, and Ian Ruchlin. Numerical generation of vector potentials from specified magnetic fields. *Journal of Computational Physics*, 379:421–437, 2019.
- [73] Kenichi Soga, E Alonso, Alba Yerro, K Kumar, and Samila Bandara. Trends in large-deformation analysis of landslide mass movements with particular emphasis on the material point method. *Géotechnique*, 66(3):248–273, 2016.
- [74] Arno Solin, Manon Kok, Niklas Wahlström, Thomas B Schön, and Simo Särkkä. Modeling and interpolation of the ambient magnetic field by gaussian processes. *IEEE Transactions on robotics*, 34(4):1112–1127, 2018.
- [75] Andrew Staniforth and Jean Côté. Semi-lagrangian integration schemes for atmospheric models—a review. *Monthly weather review*, 119(9):2206–2223, 1991.
- [76] Michael Steffen, Robert M. Kirby, and Martin Berzins. Analysis and reduction of quadrature errors in the material point method (mpm). *International Journal for Numerical Methods in Engineering*, 76(6):922–948, 2008. doi: <https://doi.org/10.1002/nme.2360>. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.2360>.
- [77] Rob Stevenson. Divergence-free wavelet bases on the hypercube: Free-slip boundary conditions, and applications for solving the instationary stokes equations. *Mathematics of computation*, 80(275):1499–1523, 2011.
- [78] Rob Stevenson. Divergence-free wavelets on the hypercube: General boundary conditions. *Constructive Approximation*, 44(2):233–267, 2016.
- [79] A. Stomakhin, C. Schroeder, L. Chai, J. Teran, and A. Selle. A material point method for snow simulation. In *ACM Transactions on Graphics (SIGGRAPH 2013)*, pages 102:1–10, 2013.
- [80] D. Sulsky, Z. Chen, and H. Schreyer. A particle method for history-dependent materials. *Comp Meth in App Mech Eng*, 118(1):179–196, 1994.
- [81] G Tóth and PL Roe. Divergence-and curl-preserving prolongation and restriction formulas. *Journal of Computational Physics*, 180(2):736–750, 2002.
- [82] Karsten Urban. *Wavelets in numerical simulation: problem adapted construction and applications*, volume 22. Springer Science & Business Media, 2002.
- [83] Ross Vennell and Rick Beatson. A divergence-free spatial interpolator for large sparse velocity data sets. *Journal of Geophysical Research: Oceans*, 114(C10), 2009.
- [84] Roberto Verzicco. Immersed boundary methods: Historical perspective and future outlook. *Annual Review of Fluid Mechanics*, 55(1):129–155, 2023.
- [85] Hongliang Wang, Roberto Agrusta, and Jeroen van Hunen. Advantages of a conservative velocity interpolation (cvi) scheme for particle-in-cell methods with application in geodynamic modeling. Technical report, Wiley Online Library, 2015.

- [86] Huayi Wei, Xuehai Huang, and Ao Li. Piecewise divergence-free nonconforming virtual elements for stokes problem in any dimensions, 2021. URL <https://arxiv.org/abs/2007.02090>.
- [87] Joshua Wolper, Yu Fang, Minchen Li, Jiecong Lu, Ming Gao, and Chenfanfu Jiang. Cd-mpm: continuum damage material point methods for dynamic fracture animation. *ACM Transactions on Graphics (TOG)*, 38(4):1–15, 2019.
- [88] Rui Xu, Peter Stansby, and Dominique Laurence. Accuracy and stability in incompressible sph (isph) based on the projection method and a new approach. *Journal of Computational Physics*, 228(18):6703–6725, 2009. ISSN 0021-9991. doi: <https://doi.org/10.1016/j.jcp.2009.05.032>. URL <https://www.sciencedirect.com/science/article/pii/S0021999109002885>.
- [89] Minglei Yang, Diego del Castillo-Negrete, Guannan Zhang, and Matthew Beidler. A divergence-free constrained magnetic field interpolation method for scattered data. *arXiv preprint arXiv:2207.12349*, 2022.
- [90] Inas Yassine and Tim McGraw. 4th order diffusion tensor interpolation with divergence and curl constrained bézier patches. In *2009 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 634–637. IEEE, 2009.
- [91] T. Ye, R. Mittal, H.S. Udaykumar, and W. Shyy. An accurate cartesian grid method for viscous incompressible flows with complex immersed boundaries. *Journal of Computational Physics*, 156(2):209–240, 1999. ISSN 0021-9991. doi: <https://doi.org/10.1006/jcph.1999.6356>. URL <https://www.sciencedirect.com/science/article/pii/S0021999199963568>.
- [92] Y. Yue, B. Smith, C. Batty, C. Zheng, and E. Grinspun. Continuum foam: a material point method for shear-dependent flows. *ACM Trans Graph*, 34(5):160:1–160:20, 2015.
- [93] Bei Zhang, Jikun Zhao, and Meng Li. The divergence-free nonconforming virtual element method for the navier–stokes problem. *Numerical Methods for Partial Differential Equations*, 39(3):1977–1995, 2023.
- [94] Fan Zhang, Xiong Zhang, Kam Yim Sze, Yong Liang, and Yan Liu. Improved incompressible material point method based on particle density correction. *International Journal of Computational Methods*, 15(07):1850061, 2018.
- [95] Xiaodi Zhang and Xiaorong Wang. A fully divergence-free finite element scheme for stationary inductionless magnetohydrodynamic equations. *Journal of Scientific Computing*, 90(2):70, 2022.
- [96] Chenxi Zhao, Yongchuan Yu, Oskar J Haidn, and Xiangyu Hu. Physics-driven complex relaxation for multi-body systems of sph method. *Computer Physics Communications*, 313:109615, 2025.