UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

VINICIUS DA COSTA DE AZEVEDO

Preserving Geometry and Topology for Fluid Flows with Thin Obstacles and Narrow Gaps

Thesis presented in partial fulfillment
of the requirements for the degree of
Doctor of Computer Science

Advisor: Prof. Dr. Manuel Menezes de Oliveira Neto

Porto Alegre
November 2016

CIP — CATALOGING-IN-PUBLICATION

Da Costa de Azevedo, Vinicius

Preserving Geometry and Topology for Fluid Flows with Thin Obstacles and Narrow Gaps / Vinicius Da Costa de Azevedo. – Porto Alegre: PPGC da UFRGS, 2016.

97 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2016. Advisor: Manuel Menezes de Oliveira Neto.

1. Fluid simulation. 2. Physics based animation. 3. Computer graphics. I. Menezes de Oliveira Neto, Manuel. II. Título.

“This planet has - or rather had - a problem, which was this: most of the people living on it were unhappy for pretty much of the time. Many solutions were suggested for this problem, but most of these were largely concerned with the movement of small green pieces of paper, which was odd because on the whole it wasn’t the small green pieces of paper that were unhappy”

— Douglas Adams

ACKNOWLEDGEMENTS

First and foremost, I would like to acknowledge my co-advisor, Christopher Batty, the Batman. Battyman is the offspring of a long successful chain of researchers in the dark art of fluid animation. A man dressed like a normal man who's able to calmly advise through enormous amounts of ignorance, never upset by slow-moving work or by my restless stubbornness. Although he has no superhuman powers (apart from those previously stated), he has keen eye for details and a honorable-code for making good research. His intellectual prowess and the ability the swiftly read student-drafts/papers make him a dangerous reviewer. All of these characteristics inspire me to become a better professional, and I will take his example for the rest of my career. I'm very grateful to you!

I'm also grateful for my advisor, prof. Manuel Menezes. His ability to craft perfect presentations and to organize and polish drafts up to the minimal details were a really valuable lesson. I also want to give a shout-out for my boys (and some girls) of UFRGS Computer Graphics/Interaction/Visualization Lab: Bernardo, Victor, Rosalia, Fred, Eduardo, Jeronimo, Andre, Guilherme, Roger, Renato, Vitor Jorge, Victor Adriel, Marcelo(s), Tales, Borja, Juliano, Lenna, Alex, Vinicius. I also grateful for the help and collaboration from my colleagues and friends of Computational Motion and Scientific Computing Group at Waterloo University: Egor, Par, Eddie, Dustin, Coleen, Ryan, Phillipe, Yipeng. Thanks everyone for the awesome moments and for the lessons learnt!

This thesis couldn't be done without the help of my family. Especially my mother, Maria Helena, which often summoned spells to help me through hard stages of my doctorate. Thanks mom, I love you! Other important family members include my step-father Sergio, my father and step-mother Mateus and Rosalia; my baby brother Anderson; my mother and sister-in-law Teresa and Elisa and my cousin Tarso.

ABSTRACT

Fluid animation methods based on Eulerian grids have long struggled to resolve flows involving narrow gaps and thin solid features. Past approaches have artificially inflated or voxelized boundaries, although this sacrifices the correct geometry and topology of the fluid domain and prevents flow through narrow regions. We present a *boundary-respecting* fluid simulator that overcomes these challenges. Our solution is to intersect the solid boundary geometry with the cells of a background regular grid to generate a topologically correct, boundary-conforming cut-cell mesh. We extend both pressure projection and velocity advection to support this enhanced grid structure. For pressure projection, we introduce a general graph-based scheme that properly preserves discrete incompressibility even in thin and topologically complex flow regions, while nevertheless yielding symmetric positive definite linear systems. For advection, we exploit polyhedral interpolation to improve the degree to which the flow conforms to irregular and possibly non-convex cell boundaries, and propose a modified PIC/FLIP advection scheme to eliminate the need to inaccurately reinitialize invalid cells that are swept over by moving boundaries. The method naturally extends the standard Eulerian fluid simulation framework, and while we focus on thin boundaries, our contributions are beneficial for volumetric solids as well. Our results demonstrate successful one-way fluid-solid coupling in the presence of thin objects and narrow flow regions even on very coarse grids.

Keywords: Fluid simulation. physics based animation. computer graphics.

Preservando geometria e topologia de escoamento de fluidos com a presença de geometrias finas e aberturas estreitas

RESUMO

Métodos tradicionais de animação de fluidos têm dificuldade em resolver escoamentos que envolvem aberturas estreitas e geometrias finas. Abordagens anteriores artificialmente inflaram ou *voxelizaram* geometrias de objetos finos, sacrificando a geometria e topologias corretas do domínio de simulação, impedindo que o escoamento interaja corretamente com regiões estreitas. No trabalho desenvolvido, apresentamos uma técnica de simulação de fluidos que respeita geometrias complexas de maneira precisa e supera obstáculos comuns em ambientes com aberturas estreitas e geometrias finas. A nossa solução baseia-se no recorte preciso de células do grid regular, gerando uma malha conformal à geometria e topologicamente correta. Nós utilizamos uma abordagem de bordas incorporadas (*cut-cells*): em cada passo do tempo, a malha de triângulos representando a superfície sólida de um objeto no domínio de simulação é recortada pelas células que intercepta, potencialmente gerando múltiplas sub-células distintas. A malha resultante é conformal ao objeto incorporado e se reduz ao grid regular em regiões que não estão em contato com a superfície. Nós estendemos as abordagens tradicionais de advecção de velocidade e projeção da pressão para dar suporte a essa estrutura de malha aprimorada. Em geral, nossa abordagem é capaz de representar melhor detalhes de geometrias que são menores que uma célula do grid, corretamente recuperando condições de contorno *no-slip* e *free-slip*, enquanto mantém uma convergência para a solução da pressão de segunda ordem no espaço. Para melhorar a advecção em regiões próximas às bordas irregulares, introduzimos um método de interpolação que funciona em células poliédricas arbitrárias, utilizando-se do método de interpolação *spherical barycentric coordinates* (SBC). Essa abordagem possibilita que as linhas características do escoamento respeitem a geometria sem penetrá-la, em contraste com métodos tradicionais de interpolação lineares ou cúbicos. Finalmente, nós melhoramos os métodos de advecção com um método FLIP modificado. Nosso método resolve uma dificuldade inerente a advecção Semi-Lagrangiana no contexto de geometrias deslocando-se através do domínio de simulação: as células que são varridas por sólidos em locomoção perdem sua informação de velocidade e tem de ser preenchidas com velocidades extrapoladas de células vizinhas. Nosso esquema FLIP garante que a informação de velocidade viaje corretamente com as superfícies, não necessitando de nenhum método de extrapolação.

Palavras-chave: Simulação de fluidos, animação baseada em física, computação gráfica.

LIST OF ABBREVIATIONS AND ACRONYMS

CGAL	Computational Geometry Algorithms Library
FLIP	Fluid-Implicit-Particle
CAD	Computer Aided Design
CFD	Computational Fluid Dynamics
PIC	Particle-in-Cell

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Current fluid animation research focuses its efforts on the visual reproduction of the complex movements of natural phenomena like water, smoke and explosions. These phenomena are pervasive in a vast spectrum of applications in computer graphics and it is important to plausibly simulate those flows in virtual environments. Through the years, several successful algorithms were proposed in computer graphics (Figure 1.1), with notable presence in films (FEDKIW; STAM; JENSEN, 2001; ENRIGHT; MARSCHNER; FEDKIW, 2002; RASMUSSEN et al., 2004; MOLEMAKER et al., 2008), animations (CHENTANEZ et al., 2007; HONG; SHINAR; FEDKIW, 2007) and interactive games (COHEN; TARIQ; GREEN, 2010; CHENTANEZ; MULLER, 2011).

Due to the complex nature of fluids and the continuous demand for high-quality special effects, fluid animation research is still an active area, despite the progresses made in the last twenty years. Some examples of highly-successful research include the unconditionally-stable advection method (STAM, 1999), non-dissipative smoke (FEDKIW; STAM; JENSEN, 2001), particle level-set for liquids (ENRIGHT; MARSCHNER; FEDKIW, 2002), particle-based fluids (MULLER; CHARYPAR; GROSS, 2003), adaptive grids (LOSASSO; GIBOU; FEDKIW, 2004), hybrid meshes (FELDMAN; O'BRIEN; KLINGNER, 2005), semi-Lagrangian surface tracking (BARGTEIL et al., 2006), variational method for fluid-solid coupling (BATTY; BERTAILS; BRIDSON, 2007), wavelet turbulence (KIM et al., 2008), Predictive-Corrective SPH (SOLENTHALER; PAJAROLA, 2009), multiscale approach for surface tension (THUREY et al., 2010), tall-cell grids (CHENTANEZ; MULLER, 2011), bubbles (BUSARYEV et al., 2012), position-based fluids (MACKLIN; MULLER, 2013), the material point method (STOMAKHIN et al., 2014), stream-function solvers (ANDO; THUEREY; WOJTAN, 2015) and Schrödinger smoke (CHERN et al., 2016).

With a given scene configuration and the nature of the fluid (e.g. water or smoke), the usual approach is to execute a numerical simulation to model plausible and consistent representations of the flow. The numerical simulation utilizes mathematical models that represent fluid movement similarly to the way they behave in the real world. The most famous model is based on the solution of the Navier-Stokes equations, which are discretized in time and space. For the space discretization, Eulerian Methods (FOSTER; METAXAS, 1997; STAM, 1999; FEDKIW; STAM; JENSEN, 2001) utilize grids of fixed points distributed on the simulation domain to evaluate fluid properties. These grids are obtained through a successive domain

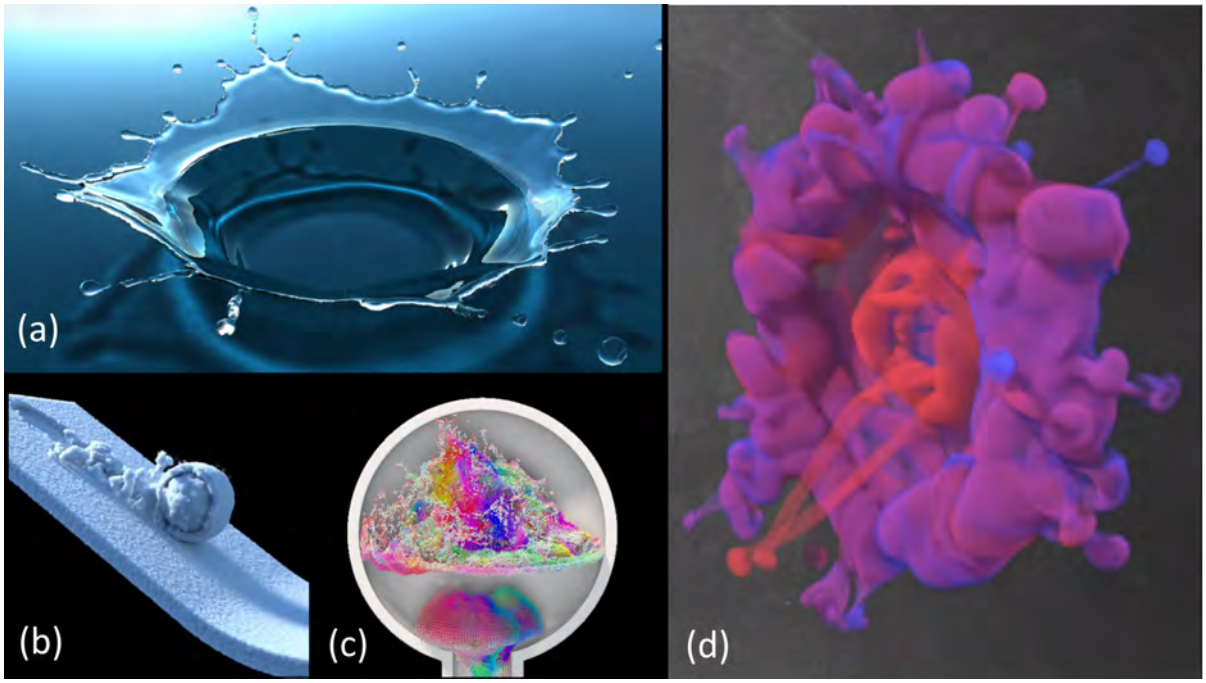


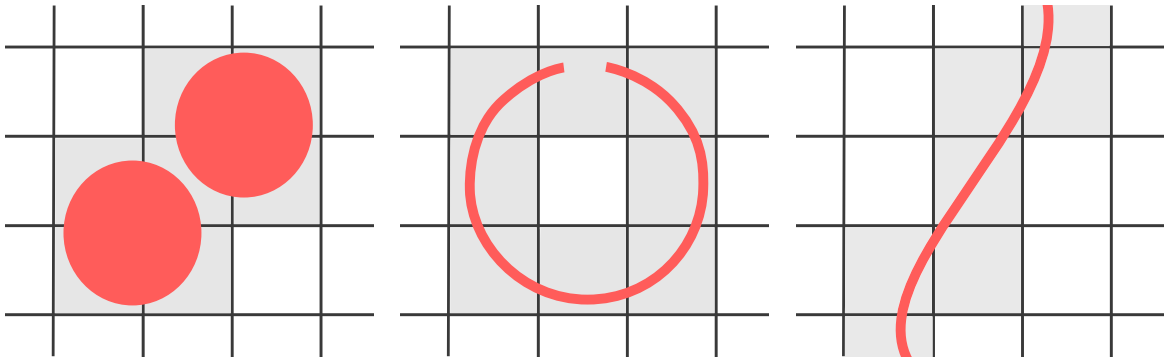
Figure 1.1: Successful fluid simulation research examples through years: (a) multiscale approach for surface tension (THUREY et al., 2010), (b) material point method for snow simulation (STOMAKHIN et al., 2013), (c) stream-function solver (ANDO; THUEREY; WOJTAN, 2015) and (d) Schrödinger Smoke (CHERN et al., 2016).

subdivision. The usual layout used on computer graphics consists of a regular structured subdivision of the domain, representing it through regular cells.

However, standard grid-based discretizations face difficulties when either the boundaries themselves or the spaces between the boundaries are thin relative to the grid resolution. For narrow flow regions, which could be caused either by spaces between objects smaller than a grid cell (Figure 1.2a) or by detailed features inherent to the geometry (Figure 1.2b), the challenge is that a typical voxelized view of the domain simply cannot capture them correctly, either topologically or geometrically. For thin boundaries (Figure 1.2c), the same difficulties are exacerbated by the need to prevent flow on one side of an impermeable boundary from erroneously interfering with flow on the opposing side.

While in principle one could continually increase the grid resolution until the thin feature or region is fully resolved, this is tremendously expensive and impractical for most animation scenarios. The poor scaling of volumetric simulation has motivated recent efforts to capture as much detail as possible at *free surface boundaries* while using much lower resolution underlying simulation grids (KIM; SONG; KO, 2009; WOJTAN et al., 2010; BOJSEN-HANSEN; WOJTAN, 2013; EDWARDS; BRIDSON, 2014). Our goal is philosophically similar: we seek to enhance the ability of coarse grid-based Eulerian fluid simulators to resolve interesting flows, but focus instead on *solid boundaries* which may be moving, irregularly shaped, arbitrarily thin, and in

close mutual proximity.



(a) (b) (c)

Figure 1.2: Different scenarios that are difficult to handle with standard regular-grid discretization: (a) narrow gap between solid objects, (b) an object with small hole and (c) infinitesimally thin object.

1.1 Thesis Statement

The central idea of this thesis is that infinitesimally thin objects, narrow gaps and complex shapes can be represented using a cut-cell data-structure that correctly models geometry and topology. Pressure and velocity samples can be connected by a graph that preserves correct topology of the domain for very coarse settings. A symmetric positive definite pressure matrix formulation can be employed to model the problem, yielding second-order convergence for pressure in space. Additionally, a carefully tailored FLIP advection scheme can be used to move boundaries and velocity information in tandem; with sufficient particle sampling, no velocity extrapolation is needed for filling-in empty spaces left by moving boundaries. Together, these concepts lend to stable and efficient simulations of flows involving thin obstacles possibly containing complex shapes and narrow gaps, relaxing a long standing restriction in fluid animation. The ability to simulate such representations on very coarse grids lends to computational savings that can be explored to efficiently create animation previews, potentially saving animators considerable time.

To demonstrate this thesis, we present an embedded boundary or *cut-cell* approach that at each time-step intersects grid cells with triangle meshes representing solid boundaries, potentially yielding multiple distinct polyhedral sub-cells per regular voxel. The resulting hybrid simulation mesh closely conforms to the geometry of the solid boundary and reduces to a regular grid away from the boundary. Crucially, and in contrast to existing fluid animation methods using regular

grids, *our approach preserves the topology of the fluid domain*, including thin solids and slender narrow gaps between nearby solids. The practical advantage this offers is a sharp reduction in the unnecessary coupling between grid resolution and solid boundary topology present in previous work; that is, fluid grid resolution can be artistically adjusted solely to achieve the desired balance of fluid detail and computational cost, without concern for whether an inaccurate solid discretization will inadvertently disconnect or merge flow regions in the process.

We first introduce a topologically-accurate, graph-based discretization for the pressure projection on the cut-cell mesh which can resolve flows in difficult regions. Furthermore, it offers greater fidelity than prior work on thin solids: it better accounts for the sub-grid geometry of the boundary, correctly recovers free-slip boundary conditions, and is consistent with existing cut-cell approaches for *volumetric* objects (e.g., (BATTY; BERTAILS; BRIDSON, 2007; NG; MIN; GIBOU, 2009)).

Secondly, to improve the handling of advection near boundaries we develop a *conforming* velocity interpolant on arbitrarily-shaped polyhedral cut-cells by relying on spherical barycentric coordinates (SBC) (LANGER; BELYAEV; SEIDEL, 2006). This allows flow characteristics to more closely respect boundary geometry than is possible with standard, boundary-oblivious linear or cubic interpolation schemes, particularly in narrow regions.

Lastly, we augment our approach with a tailored PIC/FLIP (ZHU; BRIDSON, 2005) advection scheme. Beyond its usual ability to reduce numerical dissipation, this resolves a lingering difficulty with semi-Lagrangian advection in the context of thin moving boundaries. Specifically, grid cells swept over by solids lack valid velocity information after semi-Lagrangian advection (GUENDELMAN et al., 2005), and must be filled back in by extrapolating from valid cells that may be arbitrarily far away. Our use of Lagrangian particles ensures that velocity data flows coherently with the boundaries themselves, so that extrapolation is not required.

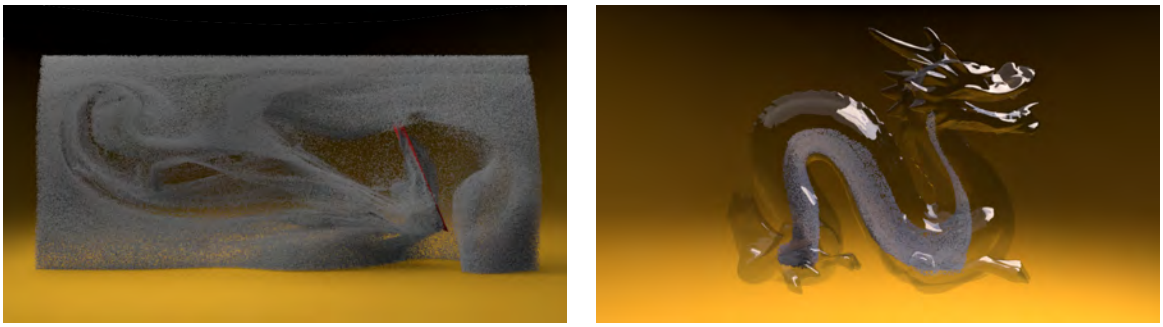
These enhancements substantially improve the detail that can be achieved when simulating fluids interacting with solid boundaries, while readily integrating into the dominant Eulerian staggered grid fluid pipeline. Figure 1.3 shows examples of fluid animations containing thin obstacles and gaps created with our technique on very coarse grids. On the left, a very thin paddle successfully stirs smoke. The image on the right shows smoke propagating through a narrow tunnel in a low-resolution grid.

To summarize, the **technical contributions** of this thesis include:

- The identification of key limitations of existing thin solid and thin gap treatments, due to voxelized geometry and standard interpolation strategies (Section 3.1);
- A symmetric, graph-based cut-cell pressure projection method that preserves the domain

topology (Section 3.2). It is the first to properly handle both thin obstacles and thin gaps between obstacles within coarse 3-D grid cells, allowing the use of less costly grids to animate flows in difficult geometries;

- An improved velocity interpolation scheme in polyhedral cut-cells based on spherical barycentric coordinates (Section 3.3), allowing flows to better respect irregular solid boundaries;
- A technique to improve velocity advection near thin moving obstacles (Section 3.4). By combining Lagrangian PIC/FLIP particles with our cut-cell scheme, velocity information is correctly propagated despite the presence of moving geometry.



$23 \times 10 \times 6$ grid $11 \times 9 \times 6$ grid

Figure 1.3: Our geometry- and topology-aware boundary treatment supports simulating smooth flows in the presence of thin solid geometry, irregular geometry and narrow gaps on *very* coarse grids.

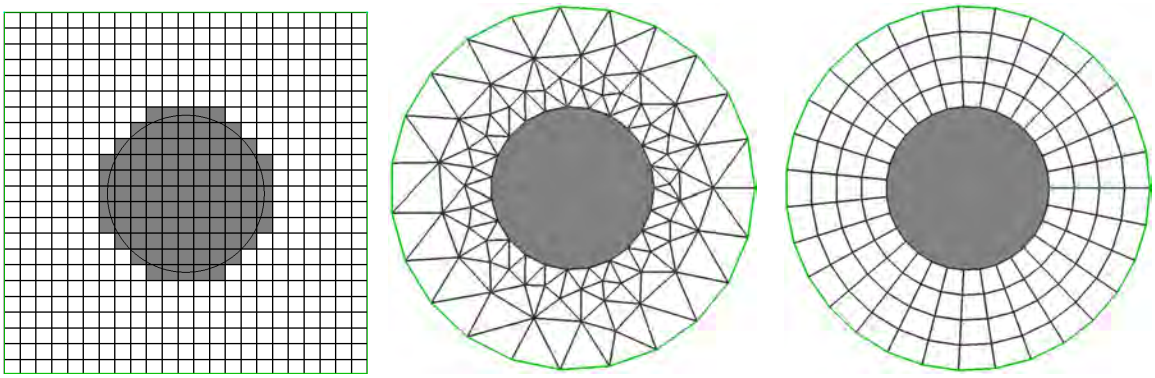
1.2 Thesis Organization

Chapter 2 discusses relevant works to this thesis, focusing on embedded meshes in regular-grid settings. Chapter 3 details the proposed cut-cell mesh structure, our topologically-aware pressure projection method, the conforming interpolation and velocity reconstruction on cut-cells, and our modified FLIP advection scheme. Chapter 4 provides specific implementation details, covering the construction of cut-cells, our least squares velocity fitting, and the implementation of Spherycal Barycentric Coordinates interpolant. Chapter 5 illustrates many results obtained with our technique, including flows in narrow regions and gaps, flows interacting with irregular shapes, and flows around thin objects. Lastly, Chapter 6 presents our conclusions and possible directions for future work.

2 RELATED WORK

Most fluid animation methods are based on the evaluation of the Navier-Stokes equations, which can be discretized in space by two popular approaches: Lagrangian, or Eulerian. *Lagrangian methods* (MULLER; CHARYPAR; GROSS, 2003; SOLENTHALER; PAJAROLA, 2009) evaluate each blob of fluid separately, so the discretized information moves along with the fluid flow. These methods can be either meshless (particles) or mesh-based; for computer graphics applications, the most common approach is to represent flows as particle systems. *Eulerian methods* (FOSTER; METAXAS, 1997; STAM, 1999), on the other hand, use fixed points distributed on the simulation domain to evaluate fluid properties. In this thesis, the proposed method is of Eulerian type. For a comprehensive review on Lagrangian fluids in computer graphics, we refer to Ihmsen et al. (IHMSEN et al., 2014).

The usual grid layout consists of a regular subdivision, discretizing the environment in a voxelized fashion. Although common and reliable, it produces rough representations for object geometry, as shown in Figure 2.1a. The resulting incorrect boundaries introduce noticeable artifacts in the simulations/animations, which may not vanish with increased grid resolution (FELDMAN; O'BRIEN; KLINGNER, 2005; BATTY; BERTAILS; BRIDSON, 2007), especially in the presence of narrow gaps. Moreover, a voxelized pressure solver still produces objectionable artifacts in thin objects settings unless high-resolution grids are used.



(a) (b) (c)

Figure 2.1: Different grid types: (a) regular grid; (b) unstructured grid; and (c) non-regular structured grid. Image from (AZEVEDO; OLIVEIRA, 2013).

Adaptive refinement techniques (LOSASSO; GIBOU; FEDKIW, 2004; LENTINE; ZHENG; FEDKIW, 2010) try to optimize the domain cell distribution concentrating fine cells near object boundaries or regions of high vorticity. While octrees (MARTIN, 1996) can improve the use of computational resources, their usage leads to non-symmetric systems in the pressure

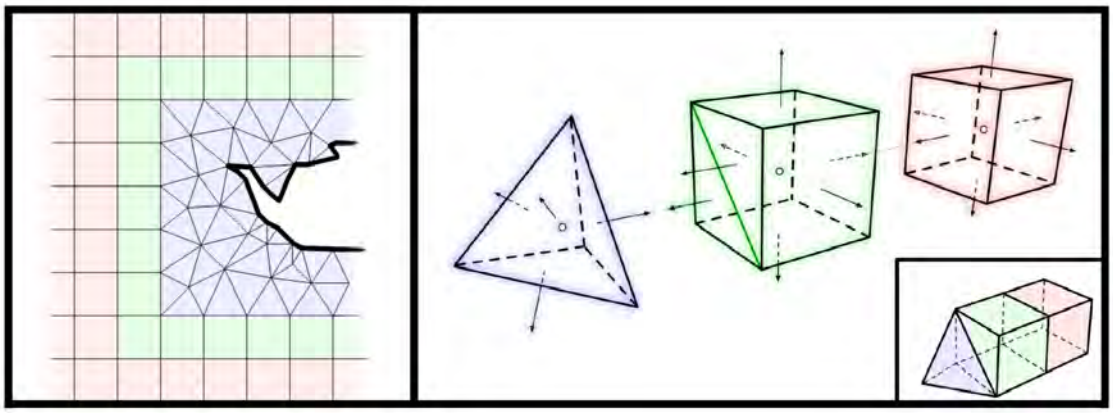


Figure 2.2: Hybrid tetrahedral meshes proposed. Left: regular cells close to object's embedded meshes are removed and replaced by triangle (tetrahedra in 3-D) meshes. On the image on the right, different types of cells in their method are shown: Tetrahedral (blue), transition (green) and regular cells (red). Adapted from (FELDMAN; O'BRIEN; KLINGNER, 2005).

equation. To overcome this, Losasso et al. (LOSASSO; GIBOU; FEDKIW, 2004) simplify the representation of the pressure gradient on T-junctions. This drops the pressure representation on those regions to first order accuracy in space, hindering the overall exactness of the method. Overlapping grids (ENGLISH et al., 2013; QIU; LU; FEDKIW, 2016) have also been used for adding details on interest regions. Adaptive refinement and boundary conforming cells are orthogonal concerns; i.e. even with very fine adaptive cells, it still will not fix errors from a voxelized boundary. Those approaches can be easily combined as in (BERGER; LEVEQUE, 1989).

Unstructured grids methods (Figure 2.1b) discretize the solution environment using triangles (2-D) or tetrahedra (3-D). These methods were introduced in computer graphics by Feldman et al. (FELDMAN; O'BRIEN; KLINGNER, 2005). Their work composed the simulation domain by fixed unstructured tetrahedral meshes and regular hexahedral cells, combining accuracy near obstacles and efficiency in open regions (Figure 2.2). It was later extended to dynamic environments (FELDMAN et al., 2005; KLINGNER et al., 2006; CHENTANEZ et al., 2007) using a mesh re-generation technique, which can take up to forty percent of the total simulation time (KLINGNER et al., 2006).

The main advantage of unstructured grids is the ability to discretize non-axis aligned boundaries with acute angles and concavities. Also, when compared with the generation of conforming structured grids, which in some cases might take weeks to custom-fit, unstructured grids generation is fast and automated. However, unstructured grids have no discernible organized structure, and node locations and neighbors need to be specified explicitly. For fluid simulation, this implies that more sophisticated and robust algorithms are required to solve the

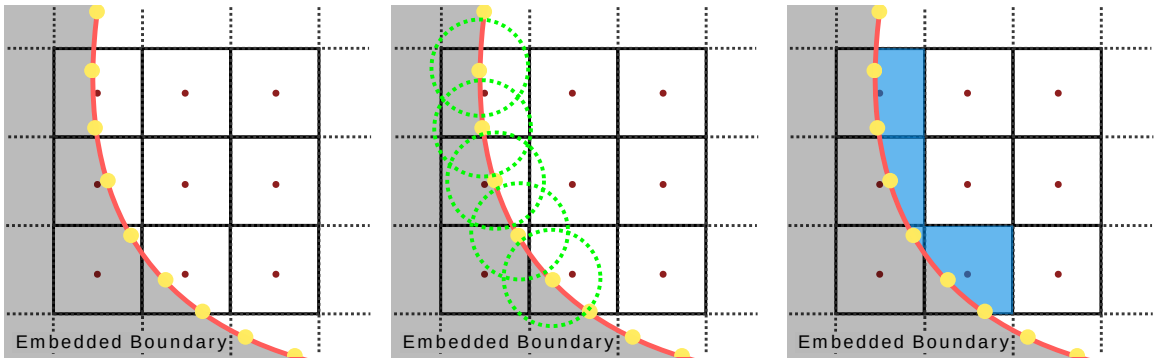
resulting system of equations, causing its solution to be slow due the overhead for accessing and preprocessing data. Also, concentrating cells in high-vorticity regions can be a complex task. This happens because triangle and tetrahedral elements do not stretch or twist well without affecting the stability and convergence of the flow solver, limiting the grid mesh to some level of isotropy. Therefore, it is often necessary to refine large portions of the grid to achieve local refinements (WYMAN, 2001).

Non-regular structured grids (Figure 2.1c) methods, also known as *curvilinear grids* (AZEVEDO; OLIVEIRA, 2013), are based on tessellations of an N-dimensional Euclidean space, adaptively contouring objects and filling the simulation domain without gaps. They are constructed using quadrilaterals (in 2-D) or hexahedra (in 3-D). Their regular structure makes the cost of the flow solver nearly identical to the ones used with regular grids. However those methods are unfeasible on dynamical 3-D scenarios due the difficulty of generating curvilinear grids on three dimensions.

In this thesis we augment current state-of-the-art for boundary treatment of Eulerian regular-grid flow solvers. Regular-grid methods are popular to computer graphics applications because no grid generation is required. Conforming non-regular grids are popular in computational fluid dynamics since the goal is to compute physically correct variables, thus models need to provide accuracy near boundaries. Our method is a combination from both worlds: our boundary cut-cells can be seen as conforming non-regular cells, though most of our grids cells remain regular. We may categorize previous embedded boundary works in two major classes (following Choi et al. (CHOI et al., 2007)): *Immersed Boundary* (Figure 2.3c) and *Cartesian Cut-cell* (Figure 2.3c) methods. Immersed Boundary methods (or diffuse interface) enforce wall conditions indirectly through the use of forcing functions, while Cartesian Cut-cells methods (or sharp interface) are based on the construction of irregular cells nearby geometry boundaries. However, this categorization is not strict, since methods from both areas often use overlapping concepts. The literature on these methods is extensive and for a thorough discussion we refer to Shelley et al. (SHELLEY; ZHANG, 2011), Hou et al. (HOU; WANG; LAYTON, 2012) and Gornak (GORNAK, 2013). On Sections 2.1 and 2.2 we will discuss important papers from both approaches.

We categorize previous work in boundary treatment for fluids simulation in two major areas (Figure 2.4): computational fluid dynamics (Sections 2.1 and 2.2) and computer graphics (Sections 2.3 and 2.4). More common to computational fluid dynamics are the cut-cell methods, which are covered in Section 2.2 of this thesis. Since these methods are more concerned with sharp representation of embedded objects inside regular grids, they are more employed in CFD

literature. More common to computer graphics, are the thin boundaries methods (Section 2.4), since usually infinitesimally thin geometries – like cloth and thin shells – are more present in computer graphics environments. Lastly, in Section 2.5, we discuss the literature about velocity reconstruction and interpolation that are relevant to this thesis. These methods are partly outside the boundary treatment for fluids simulation box, since they are widely employed in other different research areas.



(a) (b) (c)

Figure 2.3: Volumetric solid embedded in a regular-grid domain. Lagrangian mesh control points are represented as yellow circles. (a) Solid immersed in a regular-grid setting; the red line represents the interface between solid and fluid. (b) Immersed boundary method transferring function schematic view: each of the Lagrangian control points transfers its influence to a circular kernel around its location (green dashed circles). (c) Schematic view of cut-cell methods: partially-filled boundary-conforming cells (blue regions) are computed to enforce proper boundary conditions.

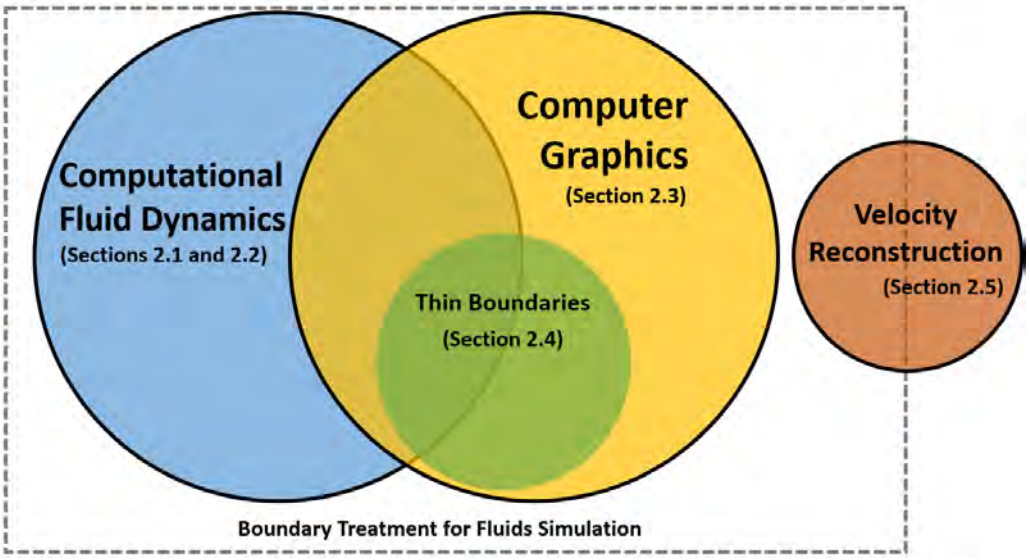


Figure 2.4: Previous works in boundary treatment for fluids simulation schematic overview.

2.1 Immersed Boundary Methods

In this section, we provide an overview of the Immersed Boundary Method. Immersed boundary methods can be mathematically modelled by continuous or discrete forcing methods (GORNAK, 2013), which will be detailed, respectively, in Sections 2.1.1 and 2.1.2.

2.1.1 Continuous Forcing Methods

Treating irregular boundaries on regular grids was first addressed by the Immersed Boundary Method (IBM) of Charles Peskin (PESKIN, 1972). Originally, the objects inside the domain were represented only by its infinitesimal boundaries modelled as elastic fibers. These fibers transferred the stress to the fluid through a continuous force field $f(\mathbf{x})$ that could be derived using constitutive laws (e.g. Hooke's law). Forces are transferred to the regular-grid flow solver as

$$f(\mathbf{x}) = \int_{\Gamma} h(s,t)\ \delta(\mathbf{x} - \mathbf{x}(s))\ ds, \tag{2.1}$$

where Γ is the embedded boundary geometry, which is parametrized by s, $h(s,t)$ is a non-linear function describing the elastic properties of the boundary in time t, δ is a delta function that maps forces to local regular-grid cells, and $\mathbf{x}(s)$ is the closest point on the boundary relative to position $\mathbf{x}$. The delta function could be replaced by a smooth distribution function to transfer the forces to grid nodes (PESKIN, 1977). IBM has been used in a variety of applications, such as simulation of blood flow in heart valves (MCQUEEN; PESKIN; YELLIN, 1982), blood clotting (FOGELSON, 1984), and aquatic animal locomotion (FAUCI; PESKIN, 1988).

Several extensions to IBM have been proposed. We highlight the virtual boundary method (GOLDSTEIN; HANDLER; SIROVICH, 1995; SAIKI; BIRINGEN, 1996) and the Immersed Interface Method (IIM) (LI; LAI, 2001; LEE; LEVEQUE, 2003). The virtual boundary method extended the forcing approach to satisfy exact no-slip boundary condition at interfaces. In this way, the force is not known a-priori, but it evolves in a feedback loop along with fluid velocities defined at the interface. Custom parameters α, β are introduced, and they account respectively for natural oscillation and dampening of the boundary response.

The Immersed Interface Method is an extension to IBM that uses boundary forces to derive proper jump conditions for pressure and velocities. IIM can achieve higher accuracy near boundaries when compared with IBM because it is able to successfully reduce the effects of the transfer function smearing due smooth kernels. The IIM is one of the most popular second-order

finite difference methods for approximating interface problems (HELLRUNG et al., 2012).

One of the disadvantages of using the immersed boundary method is that it is hard to enforce rigidity on constitutive forces that are based on the Hooke's law. This happens because strain material coefficients have to be tuned to simulate rigid bodies, leading to stiff systems.

2.1.2 Discrete Forcing Methods

Discrete forcing methods, introduced by Mohd-Yusof (MOHD-YUSOF, 1997), overcome the limitations from continuous forcing by imposing the desired value of the velocity directly, without an evolving dynamical process. Assuming discrete-time integration, we can define a velocity function on the embedded boundary as

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} = \text{RHS}^{n+1/2} + \mathbf{f}^{n+1/2}, \tag{2.2}$$

where $\mathbf{u}^n$ and $\mathbf{u}^{n+1}$ are the velocities at the beginning and at the end of the time-step, respectively; RHS is the convective, diffusive and pressure gradient terms of the Navier-Stokes equations at an intermediate step, and $\mathbf{f}^{n+1/2}$ is the intermediate forcing function. The value of $\mathbf{f}^{n+1/2}$ which will yield $\mathbf{u}^{n+1} = \mathbf{v}^{n+1}$, where $\mathbf{v}^{n+1}$ is a desired boundary velocity, is given by:

$$\mathbf{f}^{n+1/2} = -\text{RHS}^{n+1/2} + \frac{\mathbf{v}^{n+1} - \mathbf{u}^n}{\Delta t} \tag{2.3}$$

Thus, the final expression for discrete forcing methods become:

$$\mathbf{f} = \begin{cases} \mathbf{u} \cdot \mathbf{u} - \nabla^2 \mathbf{u} + \nabla p + \frac{1}{\Delta t}(\mathbf{v^{n+1}} - \mathbf{u^n}), & \text{in } \Gamma \\ 0, & \text{otherwise.} \end{cases} \tag{2.4}$$

Due its simplicity and efficiency, most of the modern embedded boundary methods use this continuous formulation to obtain a discrete form of force transferring. The difference between the various discrete forcing transferring methods is on the discretization of Equation (2.4), since the objects boundaries and the grid cells often do not coincide. Fadlun et al. (FADLUN et al., 2000) categorized three major schemes for discretizing Equation (2.4):

- Project irregular boundaries on grid faces and apply forcing conditions on the voxelized boundary, as in the lumped-mass method of Robinson-Mosher et al. (ROBINSON-MOSHER et al., 2008);

- Assume partially-filled cells to transfer forces using fractional cell formulation, as in the sharp interface immersed boundary method of Mittal et al. (MITTAL et al., 2008);
- Compute velocities on closest fluid points relative to the boundary by a special interpolation scheme, as in (KIM; KIM; CHOI, 2001).

Another set of techniques that are based on direct forcing are the ghost-cell or ghost-fluid approaches. The ghost-fluid approach was originally proposed for enforcing jump conditions for multi-phase flows in regular grids (FEDKIW et al., 1999; LIU; FEDKIW; KANG, 2000). A ghost-cell has some portion of the fluid, but its centroid is covered by the embedded boundary, which could be solid or air in liquid-air interfaces. On the other hand, a partially-filled fluid cell has some portion of the embedded boundary, but its centroid is covered by fluid. Gibou et al. (GIBOU et al., 2002) proposed a second-order accurate Poisson solver for irregular embedded boundaries by adopting ghost-cells with extrapolation to properly define jump conditions. Enright et al. (ENRIGHT et al., 2003) extended this method with a particle level-set to get second-order accurate pressures for free surface flows.

2.2 Cut-Cell methods

Cut-cell methods firstly appeared in the early work of Purvis and Burkhalter (PURVIS; BURKHALTER, 1979): the authors solved a nonlinear potential equation to predict critical Mach numbers of store configurations. Clark et al. (CLARKE; HASSAN; SALAS, 1986) solved the Euler equations on partially filled cells discretized with the finite volume method; Berger and Leveque (BERGER; LEVEQUE, 1989) and later Zeeuw and Powell (DEZEEUW; POWELL, 1993) extended this approach to support adaptivity. Quirk (QUIRK, 1994) extended previous adaptive approaches to model shock dynamics. Udaykumar et al. (UDAYKUMAR; SHYY; RAO, 1996; UDAYKUMAR et al., 1997) used cut-cells to capture free interfaces in multi-phase flows using a finite-difference scheme. Cut-cells are often modified to deal with time-step restrictions, since non-linear convective terms have to be evaluated in possibly small cut-cells. Therefore, merging small cut-cells with their adjacent neighbors is a commonplace technique in CFD methods. This is unnecessary in our approach, since we adopt a modified FLIP advection scheme that avoids this limitation (Section 3.4).

Udaykumar et al. (UDAYKUMAR et al., 2001) extended previous sharp-interface approaches for dealing with moving boundaries. The advection-diffusion terms are modified on boundary cut-cells to enforce proper velocities. The authors also dealt with the problem of

"freshly-cleared" cells, which arise when computational points that were inside a solid emerge as fluid on the subsequent time-steps. This could be treated with proper jump conditions in time for the convection equation; however, no physically-based condition can be derived for fluid-structure interaction problems. To overcome that, they merge freshly-cleared cells with their neighbors and the new velocity is computed on merged cells by a simple interpolation using nearby velocities samples. We do not observe problems created by moving boundaries in our method, since our modified FLIP advection scheme is able to move velocities information along with moving boundaries and no velocity interpolation is needed. We will further detail our modified FLIP advection scheme in Section 3.4.

Colella and collaborators (JOHANSEN; COLELLA, 1998; SCHWARTZ et al., 2006) developed a cut-cell method that interpolates velocities to lie at the centroids of partial faces. This achieves second-order accurate *velocities* at the expense of more complex stencils; however, these stencils yield non-symmetric systems and cannot be applied in narrow regions. Day et al. presented an interesting partial extension of this idea to thin boundaries in two dimensions, through the use of a more general graph structure and extra ghost samples on the grid (DAY et al., 1998). Ng et al. (NG; MIN; GIBOU, 2009) showed that the finite volume variant of Batty et al. (BATTY; BERTAILS; BRIDSON, 2007) yields second-order accurate pressures and first-order velocities. In essence, *our pressure-solve discretization applies and generalizes the work of Ng et al. to thin boundaries and regions*.

Cut-cells were also employed for achieving better convergence rates for algebraic multi-grid methods. Crockett et al. (CROCKETT; COLELLA; GRAVES, 2011) discussed the related idea of “multi-cells" which arise during coarsening steps of a multigrid scheme for Poisson problems on irregular domains. Hellrung et al. (HELLRUNG et al., 2012) presented a more complex virtual node discretization for 3-D Poisson problems with discontinuities, assuming a level-set description of domain boundaries which effectively restricts the method to closed regions that do not possess thin boundaries or gaps. They presented a family of multigrid algorithms that solve the Poisson equation with near-optimal efficiency.

2.3 Boundary Treatment in Computer Graphics

Foster and Fedkiw (FOSTER; FEDKIW, 2001) tried to minimize voxelization artifacts using the obstacle’s normal vectors to enforce appropriate velocity boundary conditions. Enright et al (ENRIGHT; MARSCHNER; FEDKIW, 2002) extrapolated velocities across the surface of the liquid to surrounding air cells to get better implicit surfaces update. This idea was

further extended by Houston et al. (HOUSTON; BOND; WIEBE, 2003) and by Rasmussen et al. (RASMUSSEN et al., 2004). Robinson-Mosher et al. (ROBINSON-MOSHER et al., 2008; ROBINSON-MOSHER; ENGLISH; FEDKIW, 2009) proposed an IBM mass-lumping scheme; their transferring function was able to better conserve torque on two-way interactions. Their approach inspired follow-up papers in CFD (ROBINSON-MOSHER; SCHROEDER; FEDKIW, 2011; QIU; FEDKIW, 2015).

A sharp interface method was introduced in computer graphics by Roble et al. (ROBLE; ZAFAR; FALT, 2005). They proposed a two-dimensional finite volume-like technique for irregular static boundaries, in which the usual Poisson stencil is augmented with per-face weights that account for the fluid fraction of each face. Batty et al. (BATTY; BERTAILS; BRIDSON, 2007) presented a closely related, variational technique that enabled stable coupling in 3-D with irregular volumetric rigid bodies; the transfer function was based on the fluid/boundary ratio of partially filled cells. Batty et al. (BATTY; XENOS; HOUSTON, 2010) further combined a sharp embedded boundary technique with unstructured grids, enabling spatially adaptive liquid simulation with more accurate enforcement of air and liquid interfaces.

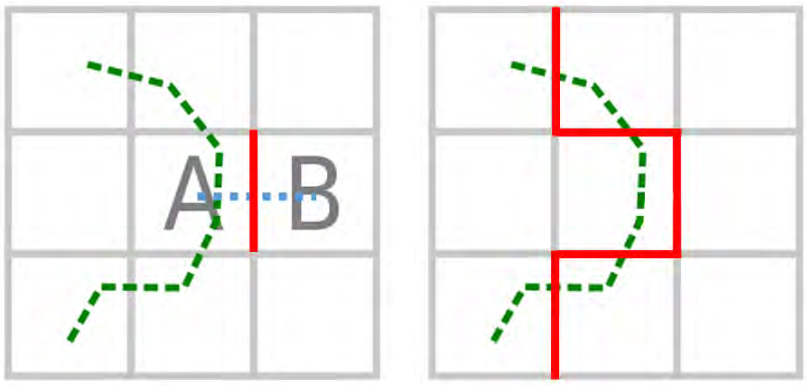
In the context of multigrid solvers, Weber et al. (WEBER et al., 2015) adapted the scheme of Ng et al., ensuring consistent discretization across grid levels, but did not consider multi-cells or the treatment of thin solids; work by Dick et al. (DICK; ROGOWSKY; WESTERMANN, 2016) is similar in spirit. Ferstl et al. (FERSTL; WESTERMANN; DICK, 2014) used a cut-cell tetrahedra-based finite-element scheme with a multigrid solver, and similarly preserved the free surface topology during coarsening, though solid boundaries were treated as voxelized.

Edwards et al. (EDWARDS; BRIDSON, 2014) proposed an adaptive discontinuous Galerkin scheme on cut-cell meshes to handle detailed free surface flow on coarse grids, potentially involving multiple disjoint liquid components per original cell; they did not discuss thin solids or thin gaps. Outside of fluid animation, topology-aware strategies have been applied to simulate the dynamics of elastic deformable objects possessing multiple distinct deforming components inside a single finite element (TERAN et al., 2005; NESME et al., 2009). Though conceptually related, they are naturally inapplicable to the problem we consider.

2.4 Thin Solid Boundaries

While thin boundaries often arise in two-way coupling, we focus our review on aspects relevant to the one-way (solid-to-fluid) coupling problem addressed by our work. The coupling of fluid to thin boundaries in computer graphics was first addressed by Guendelman et

al. (GUENDELMAN et al., 2005). Their approach voxelized the geometry of thin shells onto the regular grid, and used a one-sided extension of trilinear interpolation based on raycasting to avoid mixing data from the opposite side of a boundary (Figure 2.5). They also proposed an extrapolation approach to fill in data for fluid regions that are swept over and invalidated by moving boundaries. Later work by Robinson-Mosher et al. (ROBINSON-MOSHER et al., 2008; ROBINSON-MOSHER; ENGLISH; FEDKIW, 2009) adopted essentially the same one-sided interpolation mechanism. A similar raycasting strategy has been applied to compressible flows in computational fluid dynamics (WANG et al., 2012).



(a) (b)

Figure 2.5: Raycasting approach for representing an embedded boundary (green dashed line): shared edges between regular cells are tagged as solid (red lines), and pressure samples adjacent to solid edges are not linked to each other. A single solid shared edge, in (a), between two regular grid cells A and B. If a ray casted from the centroid of A to centroid of B (blue dashed line) intersects the embedded geometry, the shared edge between these cells is tagged as solid. The procedure is performed for all cells in (b), and the embedded object geometry is represented by the red regular grid edges.

Robinson-Mosher et al. (ROBINSON-MOSHER et al., 2008) used a mass-lumping technique for two-way coupling of thin shells to fluid on a regular grid. This sacrifices free-slip velocities even in the inviscid limit, so the same authors proposed the use of ghost-velocities and a constraint-based formulation to restrict only the normal component of velocity (ROBINSON-MOSHER; ENGLISH; FEDKIW, 2009). Both methods use a voxelized boundary approximation, and thus the topology of the fluid domain used by the pressure solver is often incorrect in tight configurations. Voxelization also leads the solid boundary velocity constraints to be applied at grid face centres rather than on the actual boundary itself. Qiu et al. (QIU; FEDKIW, 2015) proposed a two-way rigid-body-fluid coupling scheme that extends the voxelized approach to thin gaps using lower-dimensional advection and extra degrees of freedom, though it does not consider thin objects.

Boundary-conforming Eulerian tetrahedral meshes (e.g., (KLINGNER et al., 2006; FELDMAN; O'BRIEN; KLINGNER, 2005; ELCOTT et al., 2007)) could *potentially* simplify the treatment of thin boundaries during pressure projection, at the cost of repeated and potentially costly remeshing, but to our knowledge this has not been explicitly considered. The closest is the work of Chentanez et al. (CHENTANEZ et al., 2006), who simulated the coupling of fluid to deformable shells of modest thickness discretized with tetrahedra using a conforming mesh approach. To reduce meshing costs for liquid animation, Chentanez et al. (CHENTANEZ et al., 2007) later relied on the efficiency of isosurface stuffing (LABELLE; SHEWCHUK, 2007); however, isosurface stuffing conforms to an approximate isosurface rather than the exact solid geometry. In general, while conforming meshes simplify the pressure projection, their use in Eulerian schemes does not inherently resolve interpolation and advection issues near thin boundaries. In contrast to Eulerian methods, purely Lagrangian methods that rely on conforming tetrahedralizations of both fluid and solid are also possible (MISZTAL et al., 2010; CLAUSEN et al., 2013), and may better avoid these issues; again, this does not appear to have been studied.

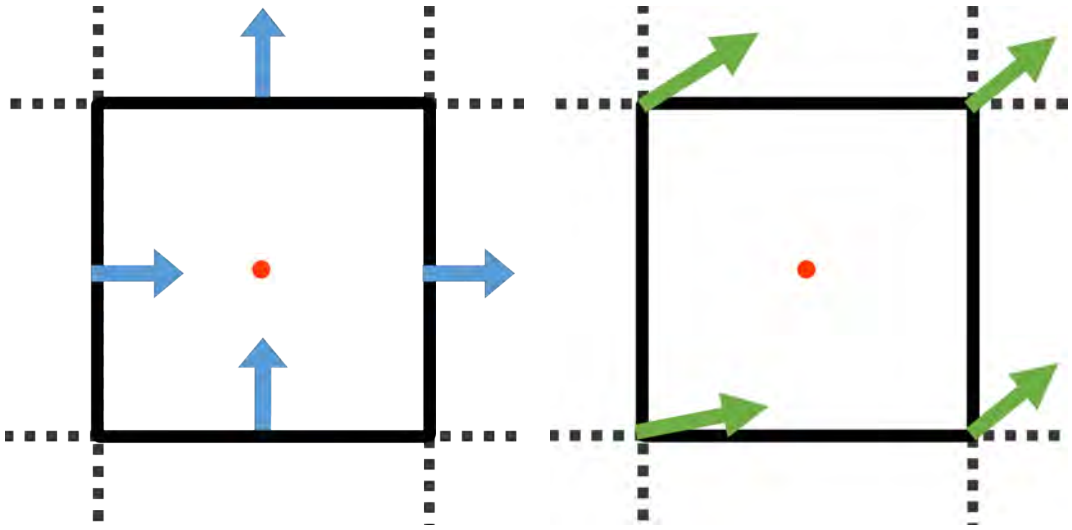
While beyond the scope of this thesis, thin objects have also been coupled to SPH simulations (e.g., (LENAERTS; DUTRÉ, 2008)). Another interesting alternative uses history-based forces to approximate the effects of fluid on submerged cloth (OZGEN et al., 2010); this does not extend to scenarios where the fluid motion itself is also of interest.

2.5 Velocity Reconstruction and Interpolation

This section reviews techniques for velocity reconstruction and interpolation. For reconstruction, the challenge is to recover all Cartesian components of the velocities at cut-cell nodal locations. For interpolation, velocities must be obtained inside cells using data that is only defined at cell boundaries. We use staggered velocities arrangement for reconstruction and for pressure projection (Figure 2.6a) and nodal based velocities for interpolation (Figure 2.6b).

2.5.1 Reconstruction

Staggered-grid projection methods recover only the face-normal components of velocity rather than full vectors; this slightly complicates interpolation and advection. In the regular grid case, interpolation can be applied on each per-axis velocity grid independently. However, for more general unstructured or polyhedral meshes, full velocities must first be reconstructed before



(a) (b)

Figure 2.6: Different velocity arrangements adopted in our method: (a) staggered velocity arrangement stores Cartesian components of the velocities (blue arrows) on the center of cell edges (faces in 3-D), while pressures (red circles) are stored on cells' centroids. In (b), nodal velocity arrangement stores all Cartesian components of the velocities (green arrows) at nodal positions.

interpolating. Velocity reconstruction uses a least squares solve to find a best-fit vector from nearby face-normal components. Feldman et al. (FELDMAN; O'BRIEN; KLINGNER, 2005) proposed to use fully mesh-free moving least squares interpolation any time pointwise velocities need to be evaluated. A more efficient method is to first recover velocities for a set of nodal points via standard least squares, and then apply a mesh-based interpolant to define the velocity over the whole domain (KLINGNER et al., 2006; ELCOTT et al., 2007). Since the number of incident faces at tetrahedral mesh vertices is often quite large, leading to overly-smooth velocity estimates, these authors also demonstrated that less dissipation is incurred by reconstructing velocities at circumcenters from just the four faces of each tetrahedron.

We use a least squares fit to recover nodal velocities from face fluxes on our polyhedral cells. Having full velocities values at the nodes simplifies the velocity interpolation used during advection.

2.5.2 Interpolation

Various velocity interpolation schemes have been proposed for use during the advection step, the most common being simple bi/tri-linear interpolation on a regular grid (STAM, 1999). Higher-order extensions have been used to improve the retention of vorticity (FEDKIW; STAM; JENSEN, 2001; SELLE et al., 2008). Guendelman et al. (GUENDELMAN et al., 2005) were

the first to directly address the interpolation issues raised by thin boundaries. Subsequent related work by Robinson-Mosher et al. (ROBINSON-MOSHER et al., 2008; ROBINSON-MOSHER; ENGLISH; FEDKIW, 2009) relied on the same interpolation technique. Raycasting is used to determine visibility between an interpolation point and the position of a velocity sample it would depend on; one-sided interpolation can then be performed using only the visible data to robustly avoid polluting the result with data from the opposite side of a thin boundary. However, since basic trilinear or tricubic interpolation do not possess knowledge of the solid position, fluid trajectories typically still cross boundaries; this necessitates the frequent use of collision-processing during advection to prevent data crossing over.

On unstructured tetrahedral meshes, the velocity reconstruction approaches discussed in the previous section are first used to determine velocities at desired nodal points; these can then be applied within a mesh-based barycentric interpolant. Given the velocities at tetrahedra centres (i.e., Voronoi vertices), generalized barycentric interpolation is applied over the convex Voronoi elements (KLINGNER et al., 2006; ELCOTT et al., 2007). Brochu et al. (BROCHU; BATTY; BRIDSON, 2010) suggested sub-dividing convex Voronoi elements up into smaller tetrahedra, and showed that this allows standard tetrahedron-based barycentric interpolation to be used with no appreciable loss of quality. Ando et al. (ANDO; THUEREY; WOJTAN, 2013) later applied a similar technique.

Polyhedral interpolants were used for the purpose of avoiding oversmoothing velocities, as compared to interpolating over tetrahedra. By contrast, our primary motivation for using polyhedral interpolation is that it enables the interpolated velocity to closely conform to the geometry of solid boundaries. Rosatti et al. (ROSATTI; CESARI; BONAVENTURA, 2005) presented a related two-dimensional technique that fits boundary-respecting linear velocity fields to the triangular, trapezoidal, and pentagonal cells resulting from the usual marching-squares cases applied to an implicit representation of the solid boundary. *Our approach clips the regular grid against the solid boundary triangle mesh, yielding arbitrary polyhedral cells*. We can then use an interpolant that handles non-convex polyhedra, i.e., spherical barycentric coordinates (LANGER; BELYAEV; SEIDEL, 2006).

2.5.3 Summary

This chapter discussed different approaches for discretizing complex geometries in fluid simulation. Our review focused on methods that adopt regular-grid configurations and we categorized embedded boundary methods by having either diffuse (immersed boundary) or sharp

(cut-cells) interface treatment. The review considered the discretization of boundary conditions in the context of computer graphics, thin boundaries, velocity reconstruction and interpolation. While several methods have been proposed for dealing individually with complicated aspects of complex geometries immersed in regular grids, our approach draws inspiration from different techniques to provide an unified framework for treating irregular shapes, thin objects and narrow gaps. Specifically, our pressure discretization applies and generalizes the work of Ng et al. (NG; MIN; GIBOU, 2009) to thin boundaries and thin gaps, achieving a symmetric positive-definite system, which supports an arbitrary number of disjoint components per cell. Finally, we reviewed several approaches that inspired our treatment of velocity reconstruction and interpolation on polyhedral cut-cells.

3 A CUT-CELL METHOD FOR HANDLING THIN OBSTACLES AND NARROW GAPS

To mathematically model our fluid solver we use the differential form of the incompressible Euler equations, which are written as

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} + \frac{1}{\rho} \nabla p = \mathbf{f} \tag{3.1}$$

and

$$\nabla \cdot \mathbf{u} = 0, \tag{3.2}$$

where $\mathbf{u}$ and p are the velocity and pressure fields, respectively, ρ is the fluid density, and $\mathbf{f}$ represents additional forces acting on the fluid. The Euler equations, which do not explicitly account for viscosity terms, are widely adopted in fluid animation (FEDKIW; STAM; JENSEN, 2001; MOLEMAKER et al., 2008; LENTINE; ZHENG; FEDKIW, 2010). The viscosity terms can be neglected since advection methods usually present some level of dissipation, which may further be re-interpreted as viscosity. Equations (3.1) and (3.2) are called the *momentum conservation* and the *mass conservation* equations.

The traditional way of solving Equations (3.1) and (3.2) is using the *projection method* (CHORIN, 1968), which consists of two main phases: (i) Advection and (ii) Pressure projection. The *advection phase* consists in estimating an intermediate value $\mathbf{u}^{*(n+1)}$ based on the velocity field $\mathbf{u}^{(n)}$ at time $t^{(n)}$, and computing external forces acting on the fluid (e.g., gravity). The *pressure projection* computes a scalar field of pseudo pressure values $p^{(n+1)}$ at time $t^{(n+1)}$ that guarantees mass conservation (i.e., enforces that $\nabla \cdot \mathbf{u} = 0$). Then, mass and momentum conservation equations are coupled by subtracting pressure gradients of $p^{(n+1)}$ from the intermediate velocity field $\mathbf{u}^{*(n+1)}$. This final step gives us the actual value of the fluid velocity $\mathbf{u}^{(n+1)}$ at time $t^{(n+1)}$.

The following sections detail the method proposed in this thesis. Section 3.1 introduces our boundary conforming cut-cells. Section 3.2 describes the proposed graph-based cut-cell pressure projection. Section 3.3 presents our interpolation method developed to maintain boundary conforming velocity fields. Finally, Section 3.4 describes our particle-based advection scheme. Implementation details will be addressed separately in the next chapter. Algorithm 3 presents an algorithmic outline of the full simulation pipeline.

Algorithm 1 Main Loop

while simulating **do**
    Advect FLIP particles (Section 3.3) and advance solid position
    Generate cut-cell mesh (Section 3.1 and Section 4.1)
    Transfer particle velocities to the mesh (Section 3.4)
    Add external forces to the mesh
    Perform pressure projection on the mesh (Section 3.2)
    Update particle velocities from the mesh (Section 3.4)
end while

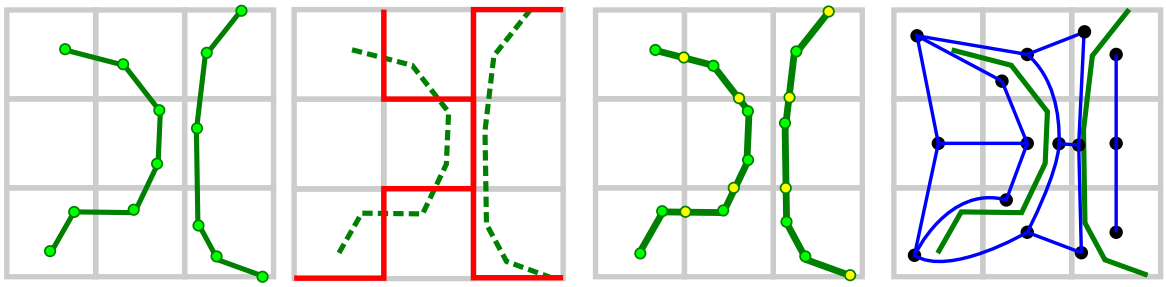
3.1 Cut-Cell Meshes

Given a triangle mesh representing the geometry of the solid boundary, we perform clipping on all cells intersected by this boundary. Each affected original grid cell may give rise to one or more boundary-conforming *polyhedral* sub-cells, which we will address simply as *cut-cells*. Clipping with triangle meshes is a well-studied problem (e.g., (AFTOSMIS; BERGER; MELTON, 1998; SIFAKIS; DER; FEDKIW, 2007; WANG et al., 2014)), most recently used by Edwards and Bridson (EDWARDS; BRIDSON, 2014) to support detailed liquid free surfaces. We will further detail our cut-cell generation algorithm in Chapter 4.

A principal difference between our cut-cell meshes and those used by Edwards and Bridson is that *we retain sub-cells on both sides of the triangle mesh geometry*. The geometry is also not required to be a “closed" surface, and therefore the triangle mesh may cut only partway through a cell. In this case, we subdivide the faces through which it crosses, but do not partition the cell itself. We will refer to the resulting faces as *dangling interior faces*. We will refer to mesh faces that connect two fluid (sub-)cells as *fluid* or *grid faces*; these will always be axis-aligned and will be inside an original regular-grid face. New faces produced by clipping against the solid boundary will be called *solid* or *geometry faces*. We do not tetrahedralize the resulting polyhedra, so cell faces may be general planar polygons. Cells that are not intersected by the geometry are left untouched, so as to be efficiently and conveniently treated with standard methods.

Figure 3.1 illustrates these cut-cell concepts in 2-D, for two infinitesimally thin solid boundaries with fluid on either side. In (a), the thin boundaries are represented by polylines, shown in green with bright green nodes. The original regular grid is shown in gray. Part (b) illustrates the boundaries (in red) resulting from the raycasting or voxelized view used by previous work (GUENDELMAN et al., 2005; ROBINSON-MOSHER et al., 2008; ROBINSON-MOSHER; ENGLISH; FEDKIW, 2009). Both the geometry and topology of the fluid domain are sacrificed: the gap between the two solid boundaries has been entirely collapsed away.

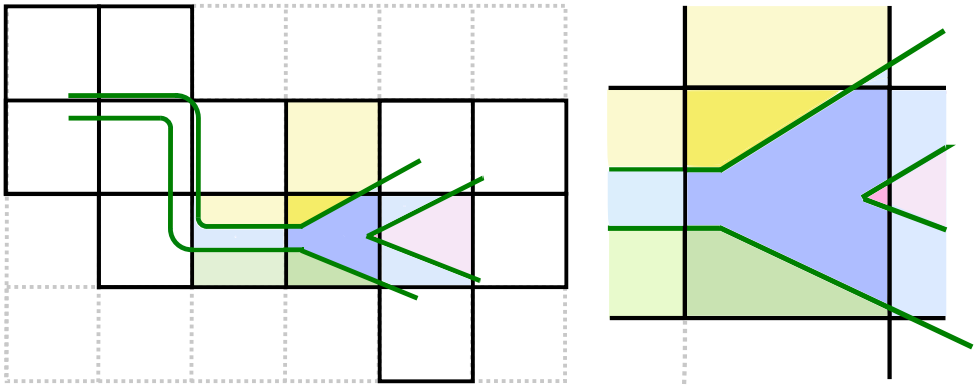
Part (c) illustrates our cut-cell mesh with the new vertices created during clipping (shown in black). Under our cut-cell view, both the thin gap and the detailed geometry of the boundary are maintained. Part (d) uses a graph (blue) to illustrate the neighbour relationships between the resulting sub-cells. The segments in the partially cut upper-left and lower-left cells are examples of *dangling interior faces*; notice that, as illustrated in the graph view, these partially cut cells are assigned only a single pressure sample although some of their faces are subdivided.



(a) Geometry (b) Raycast (c) Cut-cell (d) Graph

Figure 3.1: (a) Sub-grid thin boundaries (green) are represented by a polyline mesh in 2-D. (b) Voxelization/raycasting yields inaccurate axis-aligned boundaries (red). (c) Clipping the grid against the solid boundary mesh instead yields a cut-cell mesh with multiple distinct sub-cells, with new mesh nodes shown in yellow. (d) The connectivity relationships between sub-cells can be visualized as a graph (blue).

Figure 3.2 demonstrates the type of scenarios we can support, including narrow tubes, sharp geometry, and multiple disjoint solid boundaries and sub-cells per regular cell. For convenience and efficiency, cells away from the solid boundary can be implemented with a standard regular grid indicated by the dashed lines. An example of a complex topological domain in 3-D using a Dragon mesh is shown in Figure 3.3.



(a) Complex geometry (b) Closeup

Figure 3.2: A more complex cut-cell geometry, featuring sharp geometry and a regular cell divided into four sub-cells (bright colors). Connected neighboring sub-cells are filled with lighter shades of the same colors.

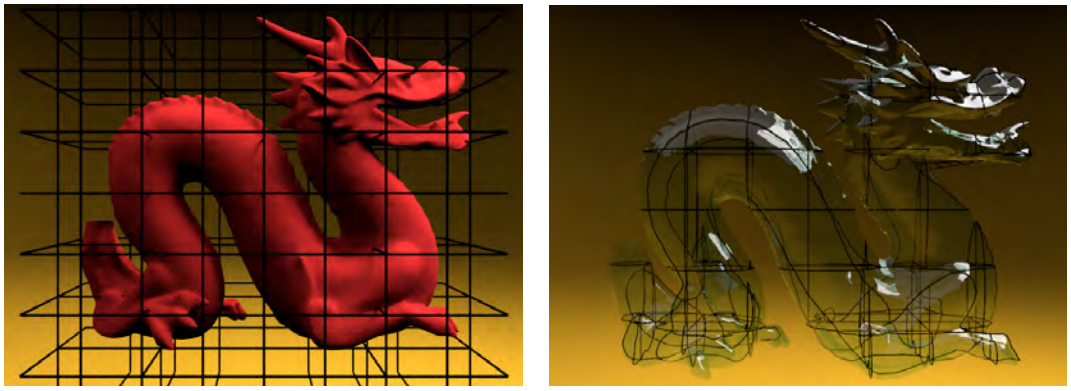


Figure 3.3: (Left) The dragon solid geometry, shown with the regular grid superimposed. (Right) The network of curves generated by intersecting the two, with the dragon rendered transparent.

3.2 Graph-Based Pressure Projection

This section details our cut-cell pressure projection algorithm. Section 3.2.1 discretize pressure gradients using standard centered differences between cell-centered pressures. Section 3.2.2 details our topology-aware pressure projection and Section 3.2.3 discusses the accuracy of the proposed approach. Finally, Section 3.2.4 presents how the proposed pressure projection algorithm deals with dangling cut-cells, which are cells that contain object's geometry with border edges or holes.

3.2.1 Cut-Cell Pressure Projection

The standard pressure projection step solves the Poisson problem $\frac{\Delta t}{\rho}\nabla \cdot \nabla p = \nabla \cdot \mathbf{u}^*$, in order to find the pressure field that will correctly convert the intermediate velocity field, $\mathbf{u}^*$, into the nearest incompressible field, $\mathbf{u}$. Having found the pressure field p, its gradient is subtracted from the velocity field: $\mathbf{u} = \mathbf{u}^* - \frac{\Delta t}{\rho}\nabla p$.

Our approach to discretizing this problem on the cut-cell mesh extends previous variational (BATTY; BERTAILS; BRIDSON, 2007) and finite volume cut-cell (ROBLE; ZAFAR; FALT, 2005; NG; MIN; GIBOU, 2009; BATTY; XENOS; HOUSTON, 2010) techniques for *volumetric* solids, which account for the flow through each face of a given grid cell adjusted for the area of the faces that are blocked by a solid obstacle. In particular, we begin with the scheme of Ng et al. (NG; MIN; GIBOU, 2009) as the basis of our approach as it *yields symmetric positive definite linear systems and pressure solutions that converge with second-order spatial*

accuracy. The associated discrete divergence measure is:

$$\nabla \cdot \mathbf{u} \approx \frac{\sum_i A_i(\mathbf{u} \cdot \mathbf{n})_i + \sum_j A_j(\mathbf{u}_{solid} \cdot \mathbf{n}_{solid})_j}{V_{cell}}, \tag{3.3}$$

where the index i runs over all fluid faces of a cell, and j runs over all solid faces. A_k indicates the area of the k-th face, $\mathbf{u}$ is the fluid velocity, $\mathbf{u}_{solid}$ is the solid velocity, $\mathbf{n}$ is the fluid face normal vector, $\mathbf{n}_{solid}$ is the solid face normal vector, and V_{cell} indicates the volume of the cell. Face normals are assumed to be oriented outwards. As usual, scaling each row of the discrete Poisson problem by its corresponding cell volume cancels volume terms in the system; we require only face areas (e.g., (LOSASSO; GIBOU; FEDKIW, 2004)). Following Guendelman et al. (GUENDELMAN et al., 2005) one must also take care to set the velocity for the solid boundary condition to be the *effective* velocity computed over the subsequent time-step rather than its instantaneous/analytical velocity, to ensure that the resulting end-of-step velocities sync with the motion of the solid during advection on the next step. The effective velocity is calculated for each node in the object's mesh by dividing its positional change by the size of the simulation time-step. In this way, particles carrying velocity information that are close to the boundary will move the exact same amount as the updated mesh. Thus, any discrepancies on the integration of mesh's position and nearby particles are avoided by using the same integration scheme.

Pressure gradients are computed using standard centered differences between *cell-centered pressures* relative to staggered velocity components, i.e. in 2-D, $\nabla p \approx \frac{p_{i+1j}-p_{ij}}{\Delta x}\mathbf{i} + \frac{p_{ij+1}-p_{ij}}{\Delta y}\mathbf{j}$, *even near cut-cell faces*. Given these discrete divergence and gradient operators, the Poisson problem can be directly discretized on the usual staggered grid. Perhaps surprisingly, Ng et al. clearly show that this projection scheme correctly converges even though the geometric centers of cells and the midpoints of faces often lie outside the actual fluid domain (see Figure 3.4, top-left). This feature conveniently preserves many of the benefits of the structured regular grid (symmetry, positive-definiteness, primal-dual orthogonality, second-order accurate pressures) as we extend it to more general topologies below.

3.2.2 Topology-Aware Pressure Projection

The method of Ng et al. implies a few restrictions. It assumes a level set description of the geometry, which limits it to volumetric objects and fluid regions larger than a grid cell width to guarantee a faithful topological description of the domain. The strictly regular underlying

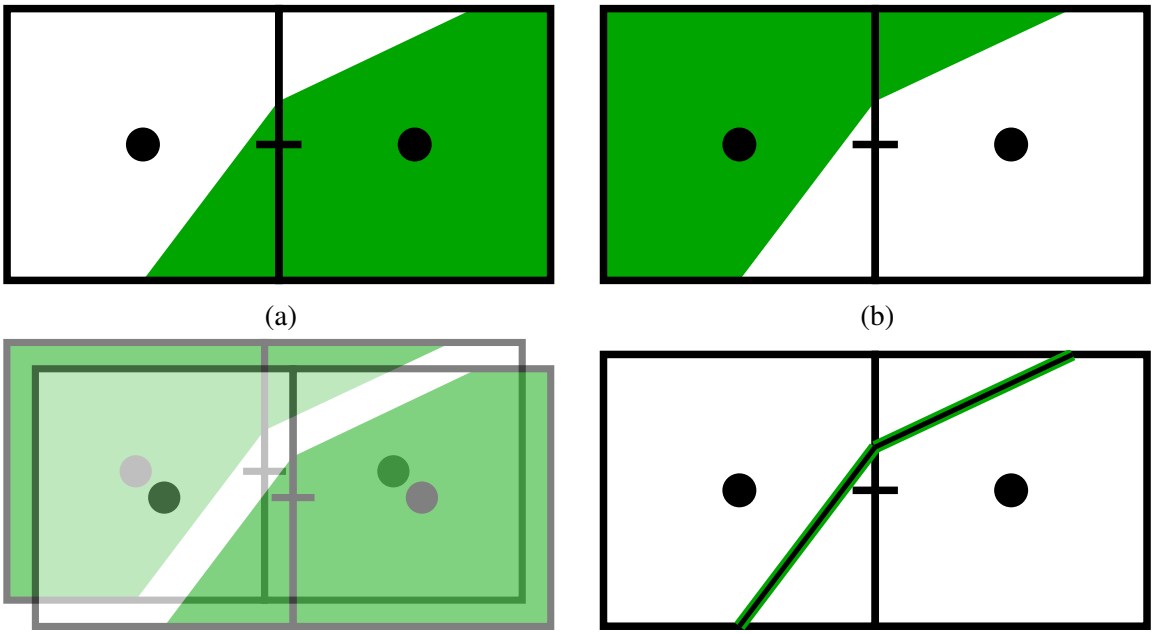


Figure 3.4: Top-left: The method of Ng et al. for embedded volumetric solid boundaries (green) converges despite using active face midpoints (black dash) and cell centers (black disks) lying outside the fluid domain (white). Top-right: A complementary dual geometry, created by swapping fluid and solid domains, can also be easily simulated with Ng's method. Bottom-left: By conceptually superimposing the top two scenarios and duplicating the required degrees of freedom, a pressure projection can be performed on the thin solid, shown at the bottom-right, without interference across it.

grid structure also means that each cell contains only a single active region and corresponding pressure. We seek to relieve these restrictions.

To extend this strategy to multiple distinct flow regions within a single regular grid cell, as produced by our mesh clipping strategy, we take inspiration from recent virtual node (MOLINO; BAO; FEDKIW, 2004; HELLRUNG et al., 2012) and topology-preserving (TERAN et al., 2005; NESME et al., 2009) schemes. We allow multiple disjoint active *sub-cells* within a single original cell, with additional pressure and velocity degrees of freedom that conceptually coincide for consistency with Ng's discretization (see Figure 3.4). We assign one pressure to each sub-cell, placing it at the original cell center's location (i.e., *not* at sub-cell centroids). Each original fluid face of the grid has multiple fluid sub-faces which connect sub-cells of adjacent regular cells together; each sub-face is assigned a velocity degree of freedom that is geometrically positioned at the regular cell face midpoint (i.e., *not* at the sub-face midpoint). This yields a more general graph structure (see Figure 3.1d) on which we can perform the pressure projection, yet the gradient and divergence operators remain axis-aligned.

In Table 3.1, we present the explicit matrix representation of our discrete Poisson equation for the small 2-D scenario shown in Figure 3.5. In large examples, most of the mesh will exhibit

the usual banded Poisson matrix structure, with a few additional unstructured entries to treat regions involving cut geometry. We also notice that pressure matrix weights (f_i on Table 3.1) on cut-cell rows only consider fluid area fractions on axis-aligned faces (edges in 2-D), ignoring any sub-cell geometry faces inside cut-cells. To calculate the matrix for the example shown in Figure 3.5, the velocities in the integral form of the divergence operator (Equation (3.3)) are replaced by $\mathbf{u} = \mathbf{u}^* - \frac{\Delta t}{\rho}\nabla p$, which yields, for a cell with indices i, j,

$$A_{i-\frac{1}{2}j}\mathbf{n}_{\mathbf{i-\frac{1}{2}j}} \cdot \left[\left(u^*_{i-\frac{1}{2}j} - \frac{\Delta t}{\rho}\frac{p_{ij} - p_{i-1j}}{\Delta x}\right)\mathbf{i}\right] + A_{i+\frac{1}{2}j}\mathbf{n}_{\mathbf{i+\frac{1}{2}j}} \cdot \left[\left(u^*_{i+\frac{1}{2}j} - \frac{\Delta t}{\rho}\frac{p_{i+1j} - p_{ij}}{\Delta x}\right)\mathbf{i}\right] +$$
$$A_{ij-\frac{1}{2}}\mathbf{n}_{\mathbf{ij-\frac{1}{2}}} \cdot \left[\left(v^*_{ij-\frac{1}{2}} - \frac{\Delta t}{\rho}\frac{p_{ij} - p_{ij-1}}{\Delta x}\right)\mathbf{j}\right] + A_{ij+\frac{1}{2}}\mathbf{n}_{\mathbf{ij+\frac{1}{2}}} \cdot \left[\left(v^*_{ij+\frac{1}{2}} - \frac{\Delta t}{\rho}\frac{p_{i+1j} - p_{ij}}{\Delta x}\right)\mathbf{j}\right] +$$
$$\sum_k A_k(\mathbf{u}^{n+1}_{solid} \cdot \mathbf{n}_{solid})_k = 0. \tag{3.4}$$

The summation over k in the equation above indicates iteration through solid faces from a cut-cell. For horizontal velocities, the dot product of the normal by the full velocity vector at an edge center becomes $\mathbf{n} \cdot (u^*\ \mathbf{i}) = sgn(\mathbf{n} \cdot \mathbf{i})\ u^*$. Applying the same procedure for vertical velocities, Equation (3.4) becomes

$$\left(\frac{\sum_k A_k p_{ij} - A_{i-\frac{1}{2}j}p_{i-1j} - A_{i+\frac{1}{2}j}p_{i+1j} - A_{ij-\frac{1}{2}}p_{ij-1} - A_{ij+\frac{1}{2}}p_{i+1j}}{\Delta x}\right) =$$
$$-\frac{\rho}{\Delta t}\left(A_{i+\frac{1}{2}j}u^*_{i+\frac{1}{2}j} - A_{i-\frac{1}{2}j}u^*_{i-\frac{1}{2}j} + A_{ij+\frac{1}{2}}v^*_{ij+\frac{1}{2}} - A_{ij-\frac{1}{2}}v^*_{ij-\frac{1}{2}} + \sum_k A_k(\mathbf{u}^{n+1}_{solid} \cdot \mathbf{n}_{solid})_k\right). \tag{3.5}$$

We can show that the system arising from the discretization on equation above is symmetric, taking advantage of a relationship between the discrete gradient and divergence operators. Specifically, considering the discrete finite-difference gradient operator G, the discrete divergence operator D can be written as $D = G^T$. Adding a diagonal matrix M, whose entries represent the fluid area fractions of each corresponding edge (left-hand side of Equation (3.5)), does not breaks this property. The system coupling pressures and velocities with matrix M is

$$\begin{pmatrix} \frac{\rho}{\Delta t}M & MG \\ G^T M & 0 \end{pmatrix}\begin{pmatrix} \mathbf{u} \\ p \end{pmatrix} = \begin{pmatrix} \frac{\rho}{\Delta t}M\mathbf{u}^* \\ 0 \end{pmatrix} \tag{3.6}$$

This system is symmetric, since the transpose of the gradient operator $(MG)^T$ is $G^T M$. The first line of the above equation yields $M\mathbf{u} + MGp = M\mathbf{u}^*$ and the second line yields $G^T M\mathbf{u} = 0$.

Multiplying first line equation by M^{-1}, rearranging it to get an expression for $\mathbf{u}$ in terms of p, and substituting in the second equation gives

$$G^T M G p = G^T M \mathbf{u}^*, \tag{3.7}$$

which represents the linear system for the pressure coupled with the velocity divergence. $G^T M G$ and $G^T M \mathbf{u}^*$ terms represent the usual B matrix and d vector (on the left and right-hand side, respectively, of Table 3.1) of the standard Poisson system $Bp = d$.

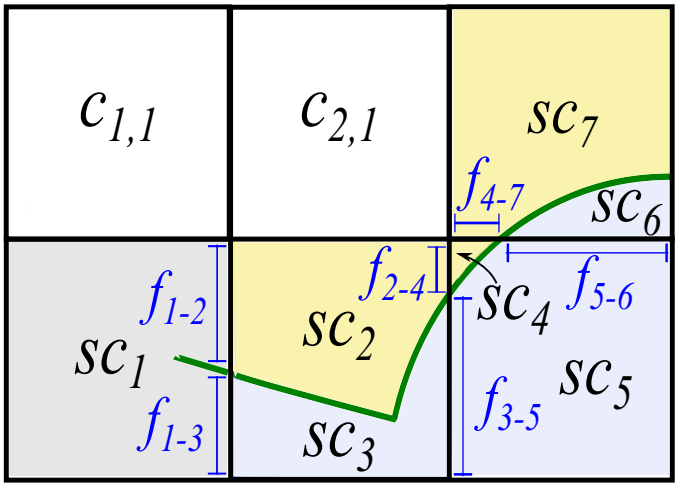


Figure 3.5: Geometry and notation used in our 2-D Poisson matrix example (Table 3.1). The solid thin boundary is shown in green. $c_{i,j}$ is a regular grid cell at row j, column i. sc_k is sub-cell k. $f_{a\text{-}b}$ is the fraction of the fluid edge shared by sub-cells a and b.

$$
\begin{array}{c}
 \\
c_{1,1} \\
c_{2,1} \\
sc_1 \\
sc_2 \\
sc_3 \\
sc_4 \\
sc_5 \\
sc_6 \\
sc_7
\end{array}
\begin{array}{c}
\begin{array}{ccccccccc}
c_{1,1} & c_{2,1} & sc_1 & sc_2 & sc_3 & sc_4 & sc_5 & sc_6 & sc_7
\end{array} \\
\left(\begin{array}{ccccccccc}
2 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 3 & 0 & -1 & 0 & 0 & 0 & 0 & -1 \\
-1 & 0 & \sum f_1 & -f_{1\text{-}2} & -f_{1\text{-}3} & 0 & 0 & 0 & 0 \\
0 & -1 & -f_{1\text{-}2} & \sum f_2 & 0 & -f_{2\text{-}4} & 0 & 0 & 0 \\
0 & 0 & -f_{1\text{-}3} & 0 & \sum f_3 & 0 & -f_{3\text{-}5} & 0 & 0 \\
0 & 0 & 0 & -f_{2\text{-}4} & 0 & \sum f_4 & 0 & 0 & -f_{4\text{-}7} \\
0 & 0 & 0 & 0 & -f_{3\text{-}5} & 0 & \sum f_5 & -f_{5\text{-}6} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -f_{5\text{-}6} & \sum f_6 & 0 \\
0 & -1 & 0 & 0 & 0 & -f_{4\text{-}7} & 0 & 0 & \sum f_7
\end{array}\right)
\end{array}
\left(\begin{array}{c}
p_{1,1} \\ p_{2,1} \\ p_{sc_1} \\ p_{sc_2} \\ p_{sc_3} \\ p_{sc_4} \\ p_{sc_5} \\ p_{sc_6} \\ p_{sc_7}
\end{array}\right)
=
\left(\begin{array}{c}
d_{1,1} \\ d_{2,1} \\ d_{sc_1} \\ d_{sc_2} \\ d_{sc_3} \\ d_{sc_4} \\ d_{sc_5} \\ d_{sc_6} \\ d_{sc_7}
\end{array}\right)
$$

Table 3.1: The symmetric cut-cell pressure projection matrix that results from the 2-D configuration shown in Figure 3.5, assuming Neumann boundary conditions on the domain perimeter. $c_{i,j}$ represents a regular grid cell at row j, column i. sc_k is sub-cell k. $f_{a\text{-}b}$ is the fraction of the fluid edge shared by sub-cells a and b. $\sum f_k$ is the sum of all fluid-edge entries in the row that represents the sub-cell sc_k. $p_{i,j}$ and $d_{i,j}$ are, respectively, the pressure and divergence at grid cell $c_{i,j}$. p_{sc_k} and d_{sc_k} are the pressure and divergence at sub-cell sc_k.

3.2.3 Primal-Dual Orthogonality

As highlighted by Batty et al. (BATTY; XENOS; HOUSTON, 2010), orthogonality of the discrete gradient estimates with respect to the face-normal velocity components is key to preserving accuracy in staggered finite volume approaches. For example, staggered octree schemes (LOSASSO; GIBOU; FEDKIW, 2004) can lose accuracy at T-junctions due to non-orthogonal gradients between large and small neighbor cells, without a more careful treatment (LOSASSO; FEDKIW; OSHER, 2005). By contrast, the gradients we use between sub-cells are always computed between the *geometric centers* of their original grid cells, and therefore preserve orthogonality with respect to the grid faces (see Figure 3.6). This property is key to both our method and that of Ng et al.

3.2.4 Dangling Cut-Cells

Our pressure discretization effectively ignores any dangling interior solid faces arising from partially cut cells, as in previous regular grid schemes for thin boundaries (e.g., (DAY et al., 1998; GUENDELMAN et al., 2005; ROBINSON-MOSHER; ENGLISH; FEDKIW, 2009)). Precisely accounting for this geometry would require generating a reasonable quality fully unstructured *conforming* mesh within the cell. While coupling a regular grid MAC scheme to a

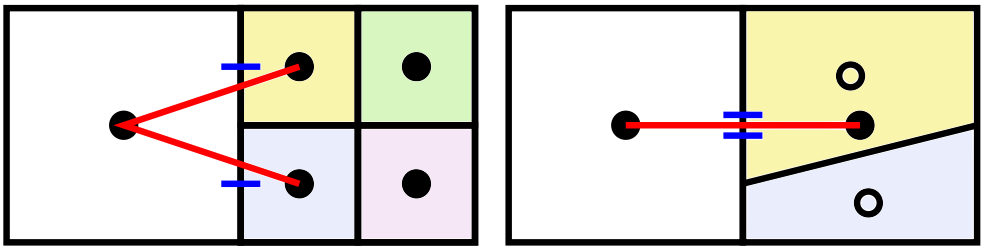


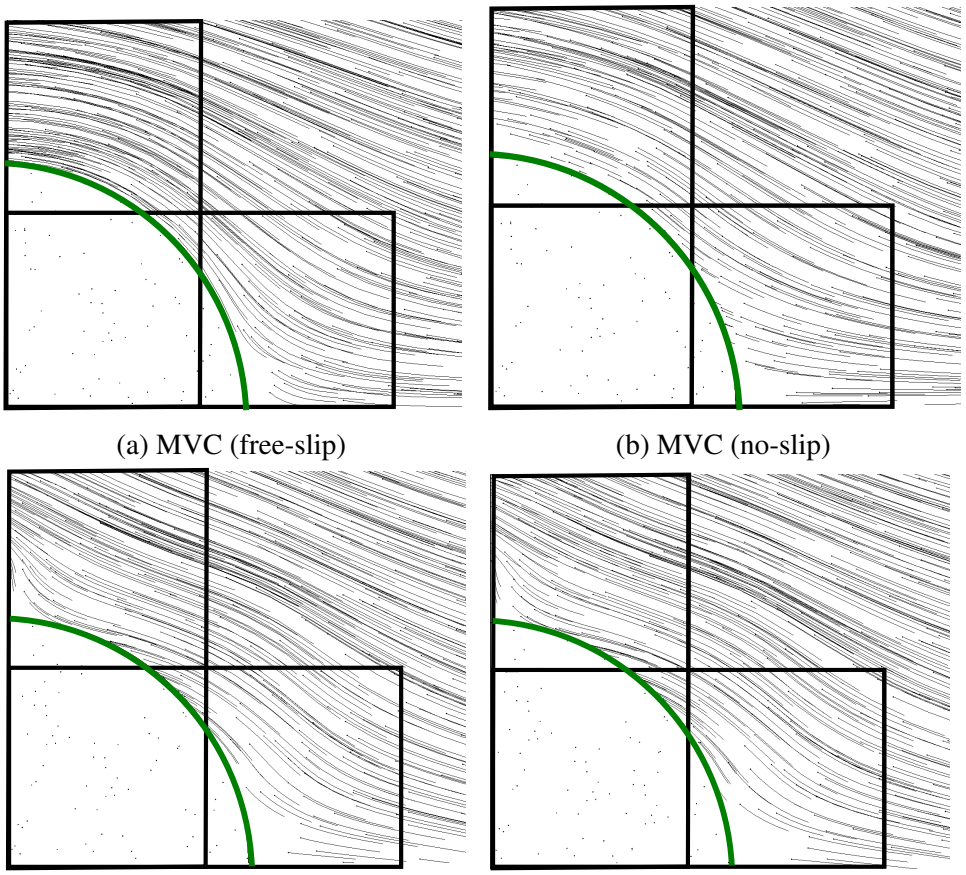
Figure 3.6: Left: Naïve octree discretizations yield face fluxes (blue) and pressure gradients (red) that are not aligned. Right: Following Ng, our cut-cell discretization co-locates all sub-cell pressures at grid cell centers (filled black circle) rather than sub-cell centroids (empty black circles). Thus our “T-junction-like" branching preserves orthogonality and avoids artifacts. Similarly, face fluxes are conceptually stored at original face midpoints (blue), rather than at sub-face midpoints.

full FEM scheme is possible (e.g., (ZHENG et al., 2015)) it sacrifices many of the benefits of our chosen *near-regular* grid discretization, both in terms of numerical properties and implementation complexity.

3.3 Conforming Interpolation on Cut-Cells

Both PIC/FLIP and semi-Lagrangian advection schemes rely on the ability to interpolate velocities at arbitrary points in the fluid domain. The interpolants used to estimate pointwise velocity values in grid-based methods typically rely on simple piecewise linear or cubic approximations (FEDKIW; STAM; JENSEN, 2001). Though effective in free flowing regions, such interpolants are fundamentally oblivious to the domain geometry, regardless of either the order of the interpolant or the accuracy of the pressure projection. As a result, the interpolated fluid velocities do not necessarily satisfy the desired no-penetration boundary condition $\mathbf{u}_{fluid} \cdot \mathbf{n} = \mathbf{u}_{solid} \cdot \mathbf{n}$, but are instead often directed *towards and through* the boundary, a fact which is particularly problematic for thin solids. The most common treatment is to apply collision detection to directly clip particle trajectories against solids (e.g., (FEDKIW; STAM; JENSEN, 2001; GUENDELMAN et al., 2005)), although this can exacerbate artificial clumping of particles and other data (RASMUSSEN et al., 2004).

We instead aim to construct an improved interpolant so that the fluid velocities themselves better respect the solid geometry, and reliance on explicit collision-processing can be reduced. Figure 3.7 uses a streamline visualization to compare one-sided bilinear interpolation (GUENDELMAN et al., 2005; ROBINSON-MOSHER; ENGLISH; FEDKIW, 2009) against our interpolation method. Although both approaches carefully avoid mixing data from the wrong



(a) MVC (free-slip) (b) MVC (no-slip)

(c) Bilinear (free-slip) (d) Bilinear (no-slip)

Figure 3.7: Streamlines of a velocity field obtained using different one-sided interpolation schemes, with a stationary flow on one side. (a) SBC interpolation from nodal velocities using *free-slip* boundary conditions leads to streamlines that flow smoothly along the boundary. (b) SBC with *no-slip* likewise conforms to the boundary, but both tangential and normal velocities drop to zero precisely at the boundary. (c) and (d) show one-sided bilinear interpolation from values stored at the regular grid corners for both free-slip and no-slip. The bilinear results exhibit grid-dependence, and do not differ appreciably from one another. Moreover, since the interpolant is non-conforming, the relevant velocity components do not drop to zero on the boundary curve.

side of the thin boundary, ours yields a velocity field that conforms more closely to the boundary, reduces grid-dependence, and can easily be set to satisfy either a free-slip or no-slip condition as desired. By contrast, the results for bilinear interpolation do not align with the boundary and exhibit nearly identical results under free-slip and no-slip conditions.

We describe our interpolation approach below, and use it during the particle advection step of Section 3.4.

3.3.1 Polyhedral Cut-Cell Interpolation

Away from solid boundaries, we apply standard trilinear interpolation to the regular grid velocities. For polyhedral (sub-)cells abutting the solid geometry, we will first reconstruct nodal velocity values (Section 3.3.2), and use these values for interpolation.

To perform interpolation over potentially non-convex polyhedra with general planar polygonal faces, we make use of spherical barycentric coordinates (SBC) (LANGER; BELYAEV; SEIDEL, 2006), which provide a convenient generalization of standard barycentric coordinates to this case. We select SBC over the more widely-known mean value coordinates (JU; SCHAEFER; WARREN, 2005), because SBC supports polygonal rather than triangular faces. This method is effective for the vast majority of cut cells, and the result is a nicely conforming interpolant.

Unfortunately, SBC cannot be readily applied to the comparatively small set of cells containing dangling interior faces, as this represents a degenerate configuration (essentially two coincident but oppositely oriented faces). The simplest way to treat this is to just thicken or extrude the input geometry into a very slim volume before simulating. This naturally eliminates problematic dangling interior faces, allowing SBC to work as usual. However, while the geometry is thin it is no longer of infinitesimal width.

The alternative is to try to construct an interpolant that is effective in the presence of the problematic dangling faces. We experimented with various approaches, including visibility-aware SPH interpolation, visibility-aware Shepard (SHEPARD, 1968) (i.e., inverse distance-weighting with a raycast check of mutual visibility) interpolation, or simply ignoring the dangling geometry altogether and reverting to trilinear interpolation similar to previous work. None of these choices is entirely satisfactory, because none ensure a velocity field that is consistent with interpolation on neighbouring cells, conforms to the interior geometry, and avoids mixing data from opposing sides of the geometry. For examples involving truly infinitesimal width geometry, we used the Shepard interpolation approach; we highlight this as an interesting challenge for future work. Our modified Shepard interpolation is represented by:

$$\mathbf{u}_S = \frac{\sum(\mathbf{u}_{n_i}/d_i) + \sum(\mathbf{u}_{n'_j}/d'_j)}{\sum(1/d_i) + \sum(1/d'_j)}, \tag{3.8}$$

where i runs over all visible nodes and j runs over all virtual nodes, d_i is the distance from S to visible node n_i and d'_j is the distance from S to virtual node n'_j. Virtual nodes are obtained by intersecting rays from S to nodes occluded from S by the solid boundary. The velocity at a virtual node is the same as the velocity of its containing boundary. Figure 3.8 (left) illustrates this

concept, where the visible nodes (n_1 and n_2) are shown in blue, and the virtual nodes (n'_3 and n'_4) are shown as yellow diamond shapes. Figure 3.8 (right) shows an example of velocity field interpolation obtained for a scenario containing a static solid boundary (in green) using free-slip. Note the smooth transitions of the vector field at the interfaces between the cell containing the dangling face and the cut cells just below it.

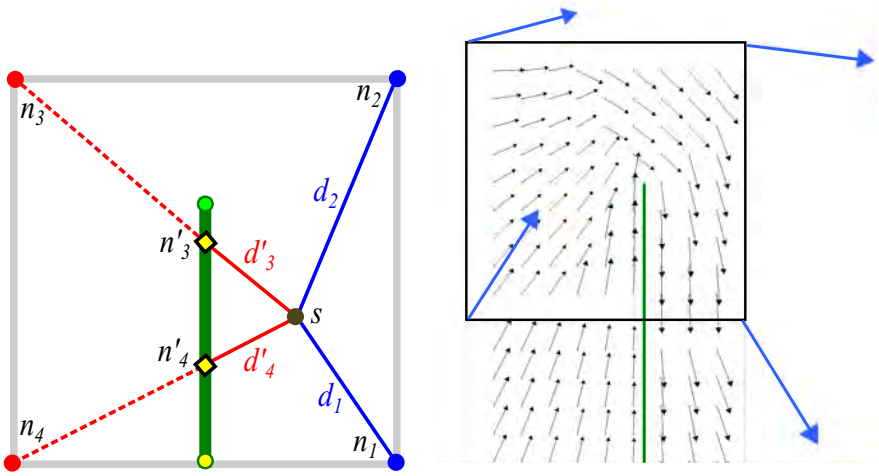
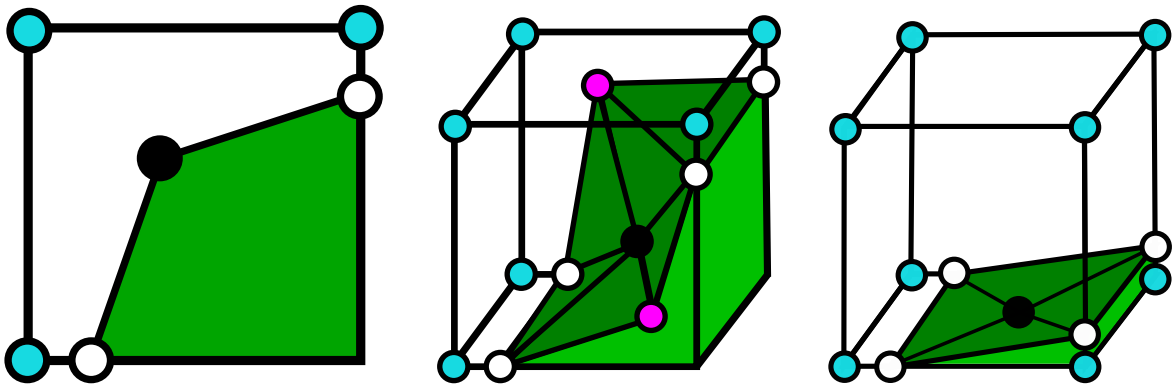


Figure 3.8: Velocity interpolation inside a cell containing a dangling interior face (green). (left) Inverse-distance interpolation at a sampling point S is performed using the velocities at the visible nodes (n_1 and n_2) , and at "virtual nodes" (n'_3 and n'_4) on the solid boundary intersected by the rays from S to the occluded nodes (n_3 and n_4). d_i is the distance from S to a visible node n_i and d'_i is the distance from S to a virtual node n'_i. (right) Example of a velocity field interpolation for a scenario containing a static solid boundary using free-slip.

3.3.2 Velocity Reconstruction

To better demonstrate cut-cell velocity reconstruction, we define three types of cut-cell nodes, illustrated in Figure 3.9. *Fluid nodes* (cyan) are the original nodes of the regular grid outside the object, for which all incident faces are axis-aligned fluid faces. *Solid nodes* (black) are nodes on the solid for which all incident faces are solid faces. *Edge-mixed nodes* (white) are nodes incident on both solid and fluid faces generated by the clipping process and lie on edges of the original grid. For 3-D, the geometry is a set of triangles (surface) rather than a piecewise linear curve. Therefore, there is one additional node type, which we call *face-mixed* nodes (magenta) and they are also added by the intersections between grid faces and the objects' meshes. Face-mixed nodes lie on the *faces* of the original 3-D regular grid, and will be treated differently on the velocity reconstruction phase.

Recovering a full velocity at fluid nodes is done by averaging from the staggered data on the incident fluid faces (edges in 2-D) for both free-slip and no-slip boundary conditions. The averaging procedure considers the distance of faces centers (edges in 2-D) to nodal locations, similar to weights obtained in inverse-distance Shepard interpolation. Other nodes will be addressed depending on the boundary condition type and will be discussed on the next sections. We consider free-slip first, in which the solid velocity determines the normal component, and fluid velocities dictate tangential components.



(a) (b) (c)

Figure 3.9: Different types of nodes for 2 (a) and 3 (b and c) dimensions. Fluid nodes (cyan) are the original nodes of the regular grid that were not intersected by the object; mixed nodes (white) are the incident on both solid and fluid faces and are on top of grid edges; face mixed nodes (magenta, 3-D only) are those that are on top of grid faces; and solid nodes (black) are the original mesh nodes that were not modified by intersections with grid planes.

3.3.2.1 Free-Slip Case

For mixed nodes, we apply a weighted least squares fit to the normal velocity components corresponding to the incident faces (solid and fluid) on the same side of the solid boundary. As in previous work (e.g., (FELDMAN; O'BRIEN; KLINGNER, 2005)), the least squares fit can be expressed as

$$\min_{\mathbf{u}} \|\mathbf{WNu} - \mathbf{Wz}\|^2, \tag{3.9}$$

where $\mathbf{W}$ is the weights matrix, $\mathbf{N}$ is a matrix whose rows are the face-normal vectors, $\mathbf{z}$ is a column vector whose entries are the corresponding face-normal velocity components, and $\mathbf{u}$ is the reconstructed velocity vector. We use the (inverse) distances from the node to the face centre as the weights; thus, the weight matrix is defined by:

$$W_{ii} = \frac{1}{r_{ii} + \epsilon}, \tag{3.10}$$

where r_{ii} is the distance of the mixed node and one of the velocities sample and ϵ is a small value to avoid division by zero. The system can occasionally be underdetermined if a mixed node has nearly co-planar faces, as shown in Figure 3.9c; however, this can be compensated for by incorporating extra face-normal velocity samples from previously unused faces from the cut-cell were the mixed node lies. We provide additional insight into our weighted least squares scheme and the addition of face-normal velocity samples in Section 4.2.

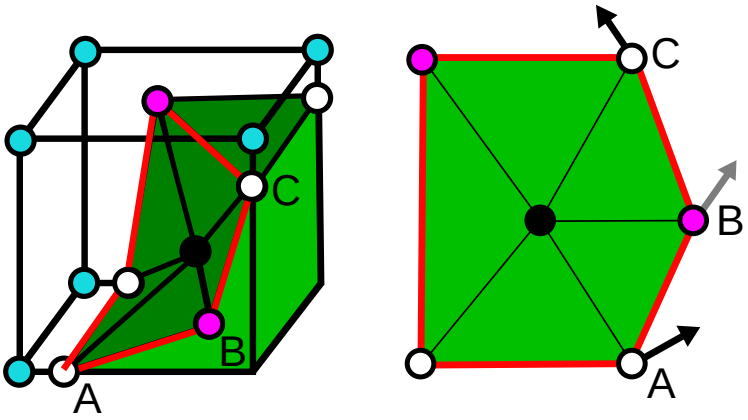


Figure 3.10: Visualization of the velocity propagation scheme. The image on the left shows a 3-D cut-cell, with three nodes tagged as "A", "B" and "C". The image on the right shows the planar view of the nodes where the velocity propagation will occur. The velocity is found on edge-mixed nodes by the use of the weighted least squares algorithm. Then, the velocity on face-mixed nodes is interpolated from nearby edge-mixed nodes (e.g., velocity on "B" is interpolated from "A" and "C"). Lastly, the velocity on mixed nodes (red edges) is iteratively propagated to inside solid nodes.

If all faces incident to a node are solid faces (e.g., the solid black node of Figure 3.9b), we extrapolate from the nearby mixed nodes for which a valid velocity has been reconstructed as described above; let us call these *valid nodes*. We call *invalid nodes* the solids nodes for which we have yet to assign a velocity. For this procedure to work, all velocities at the outer boundary mixed nodes have to be found before the propagation to the internal solid nodes can be made. In 2-D, we perform this extrapolation by simply linear interpolating along the solid boundary curve inside the cell, but considering the manifold distance between vertices (Equation (3.11)) on the original's object geometry. Initially, edge-mixed ones are the only nodes incident to solid faces that are marked as valid.

Starting from reconstructed velocity data at edge-mixed nodes, we linearly interpolate along the solid boundary curves so that every face-mixed node on the boundary curve has valid data. Figure 3.10 (right) represents the topological graph for the 3-D example shown Figure 3.10 (left). In this example, the velocity for the face-mixed "B" node is defined by a

linear interpolation between "A" and "C" edge-mixed nodes, similarly to the 2-D approach, but considering the distances on the surface manifold. That is, the velocity on "B" is calculated by:

$$\mathbf{u_B} = \frac{||p_A - p_B||\mathbf{u_A} + ||p_B - p_C||\mathbf{u_C}}{||p_A - p_B|| + ||p_B - p_C||} \tag{3.11}$$

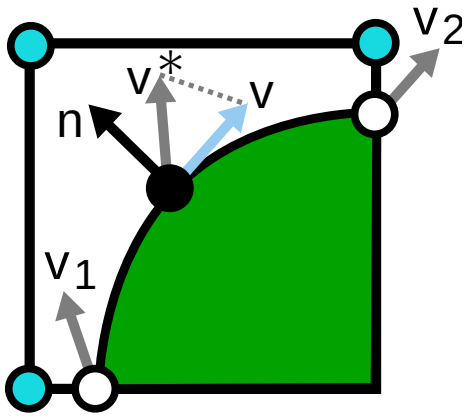
We label the internal (solid) nodes of a surface patch to be initially invalid, and perform a simple iterative averaging approach to extrapolate into the interior solid geometry: all invalid nodes with a valid neighbour are set to the arithmetic mean of the valid neighbours, and then marked as valid. This process is iterated until no invalid nodes are left. Once all interior nodes have valid data assigned, we perform a few additional iterations of repeated averaging to smooth the velocities towards a steady distribution. To avoid the damping of velocities introduced by this averaging process, we perform this step for vector magnitudes and directions separately, re-combining them at the end.

We further improve the degree to which the interpolated velocity remains tangent to the solid by directly projecting out the normal component (Figure 3.11) of the relative velocity between the solid and the fluid for mixed and solid nodes (similar in spirit to *constrained velocity extrapolation* (HOUSTON; BOND; WIEBE, 2003; RASMUSSEN et al., 2004)). This amounts to computing a new fluid velocity as

$$\mathbf{u}'_{fluid} = \mathbf{u}_{fluid} - ((\mathbf{u}_{fluid} - \mathbf{u}_{solid}) \cdot \mathbf{n}_{solid})\mathbf{n}_{solid}. \tag{3.12}$$

We observe that even if nodal velocities are projected to be orthogonal to vertex normals, the interpolated flow may *still* clearly cross boundaries if the geometry is sharply concave, i.e., a interpolated velocity may still have trajectories that do not satisfy $\mathbf{u}_{fluid} \cdot \mathbf{n} = \mathbf{u}_{solid} \cdot \mathbf{n}$ at all points along the perimeter of the cell. In 2-D, we further handle this by treating sharp corners with a no-slip condition, and smooth curves with the preceding approach; this can be observed in our results. However, in 3-D such a treatment is not straightforward, and we have not pursued it.

We may not be able to reconstruct full velocities at face-mixed nodes, as they may lack enough data to reconstruct a full 3-D fluid velocity. However, there are cases where a boundary loop consists of only face-mixed nodes, and no edge-mixed nodes, e.g. a cylindrical tube with diameter less than a grid cell width cutting horizontally through a cell face (see Section 4.1.5.1 for more details). The face-mixed nodes on the boundary of the cylinder have only one fluid face component, and the solid normal contribution may only reliably determine one of the remaining two axes. At present, we are therefore limited to scenarios in which each such a solid geometry



(a)

Figure 3.11: Projection of normal components of the velocities on cut-cell nodes. Edge-mixed node velocities $\mathbf{v}_1$ and $\mathbf{v}_2$ are found by the weighted least squares method. The unprojected velocity $\mathbf{v}^*$ is interpolated from $\mathbf{v}_1$ and $\mathbf{v}_2$. The normal of the solid node is represented as $\mathbf{n}$ and the projected velocity $\mathbf{v}$ is orthogonal to it.

patch includes at least one edge-mixed node on its boundary. A reasonable approach in severely under-resolved cases would be to reconstruct the available dimensions, and set the the remaining dimension to zero. In the cylindrical example, the missing dimension would correspond to circular rotations around the cylinder's dominant axis, which is not provided by the fluxes along the axis or the solid normal velocities perpendicular to it.

3.3.2.2 No-Slip Case

If no-slip interpolation is preferred for visual purposes, fluid nodes are again treated using least squares, but all mixed and solid nodes are directly assigned the solid velocity, including its tangential component. However, note that no-slip conditions will tend to rapidly damp out relative tangential flow in very slender gaps. This is because fluid nodes are the only nodes that no-slip conditions do not effect, and narrow gaps may contain relatively few such nodes. Hence, velocities in these sub-cells will be totally dominated by the solid, and will halt the flow entirely near static solids; an example can be seen in Figure 3.12 (bottom). No-slip can also lead to extra particle clumping, because it is fundamentally inconsistent with the inviscid free-slip condition inherent in the pressure solve.

Figure 3.12 illustrates the case of parallel line segments aligned with the flow direction. Free-slip (top) allows the flow to pass undisturbed, while no-slip (bottom) causes some drag and deflection in the velocity field. For free-slip, no velocity dissipation happens, regardless of the number of parallel lines per cell, as long as a sufficient number of particles is available (Figure 3.12 top). Table 6.1 shows, for a fixed 2D grid resolution (16×10), performance figures

for 1, 8, and 64 lines per cell over 5 cells (Figure 3.12 top). In such a scenario, as the percentage of cut cells relative to the regular 160 grid cells increases from 5% to 123.12% $(24.62\times)$, the cut-cell generation time increases $38\times$. Even for the 64-line case, the total time is still dominated by pressure projection.

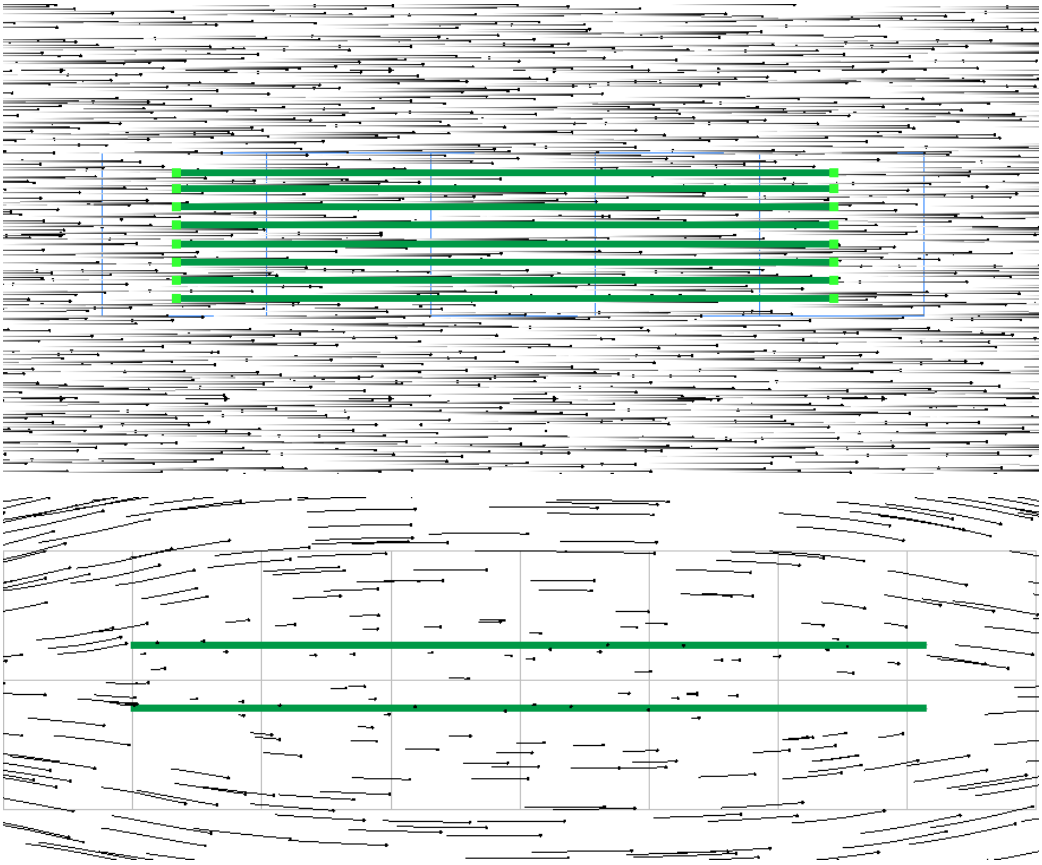


Figure 3.12: Horizontal straight segments aligned with the flow. (Top) Free-slip allows the flow to continue undisturbed. (Bottom) No-slip causes drag on the fluid and a deflection in the flow.

3.4 Fluid-Implicit-Particle Advection

The Fluid-Implicit Particle Advection (FLIP) and Particle-In-Cell methods (PIC) were popularized in computer graphics by the work of Zhu and Bridson (ZHU; BRIDSON, 2005). The authors track fluid quantities in regular grids with an explicit particle-based representation, contrary to the traditional Semi-Lagrangian method, which creates particles at velocity locations that are deleted after the advection time-step ends.

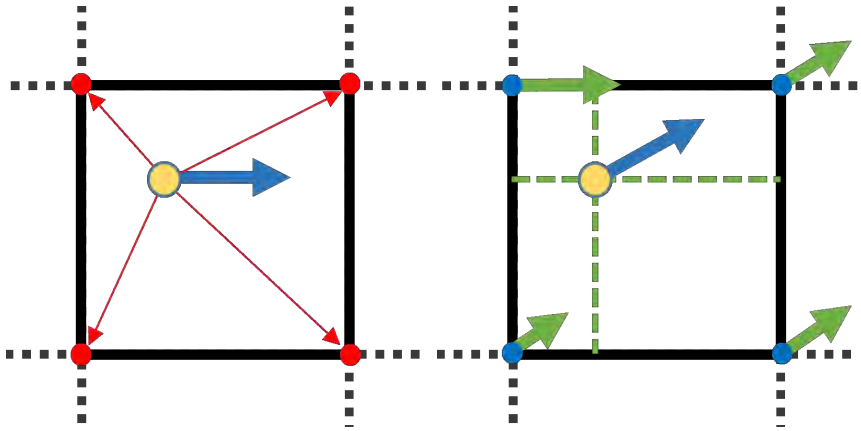
Algorithm 2 PIC/FLIP velocity update

```
if particle inside cut-cell then
    Integrate particles position with forward Euler
else
    Integrate particles position with Runge-Kutta-2
end if
(External step: solid position is updated by the object's mesh integrator)
Transfer velocities from particles to simulation mesh nodes
if FLIP velocity update then
    Store intermediary velocity field
end if
(External step: mass conservation is enforced through projection of velocity)
if PIC velocity update or particle inside cut-cell then
    Interpolate velocities and transfer to particles
else
    Interpolate velocities difference and transfer to particles
end if
```

The conventional FLIP/PIC advection scheme starts by integrating all particles positions using a specified integration scheme. Then it transfers velocities information from particles to simulation mesh nodes (Figure 3.13a). This step creates an intermediary velocity field which will be projected in its divergence free part by the pressure projection step. Depending if one chooses FLIP or PIC, the intermediary velocity field is stored. Finally, all particles velocities are updated, and information goes from grid nodes to the particles (Figure 3.13b); the difference between FLIP and PIC is on the way that velocities are updated in this step. Algorithm 2 summarizes our FLIP/PIC method, which is slightly modified to work with cut-cells. Steps inside parenthesis are evaluated by external simulation functions and will not be covered in this Section. All other steps of the above algorithm are detailed on the next subsections.

3.4.1 Integration of Particle Positions

Particle positions are updated in our method in different ways, depending if they are inside cut-cells or not. Runge-Kutta methods integrate ordinary differential equations using intermediary steps for improved accuracy and stability. While this can be successfully employed in the bulk of the fluid, these intermediary steps cannot be efficiently evaluated inside cut-cells. This happens because evaluating velocities in fractional time-steps would require intermediary cut-cells configurations, which might increase the computational time disproportionally to the gain in quality. Therefore, we chose to use forward Euler integration on particles that are inside



(a) (b)

Figure 3.13: Two distinct FLIP advection operations: (a) particle-to-grid (transfer) and (b) grid-to-particle (interpolation). The particle-to-grid operation transfers velocities from particles to grid nodes. The transferring function is discretized with a SPH kernel spiky kernel (Equation (3.13)). The grid-to-particle operation is a interpolation from the grid velocities to the evaluated particle. This is performed using a bilinear (trilinear in 3-D) interpolant on regular cells, and a Spherical Barycentric Coordinates interpolant on cut-cells.

cut-cells. This approach yields correct results, since particles that are close to moving geometries will have the same effective velocity encountered at boundaries. For the majority of the fluid, we use a standard midpoint Runge-Kutta-2 integration scheme.

3.4.2 Transfer to Mesh

We transfer velocity information carried by the particles onto the simulation mesh nodes by setting each nodal velocity to be a weighted average of the velocity of all particles in the cells incident on that node. We use SPH kernels throughout, with one minor twist in presence of solid geometry. We use a raycast to check whether the node in question is visible from the particle, and if not, we discard its contribution to that node. When face-normal velocity components are needed by the pressure solve, they are computed by interpolating the velocity vector at face's centroid from cell nodes and taking the dot-product with the corresponding face normal. We use a SPH kernel radius of twice the grid cell width, although one can safely use smaller kernels provided they cover one full cell. Our kernel of choice for transferring the velocities from particles to the grid is the spiky SPH kernel (MULLER; CHARYPAR; GROSS, 2003)

$$k_{\text{spiky}}(r,h) = \frac{15}{2\pi h^6} \begin{cases} (h-r)^3, & 0 \leq r \leq h \\ 0, & \text{otherwise,} \end{cases} \tag{3.13}$$

where r is the the distance from the particle to the grid velocity sample and h is the kernel size.

3.4.3 Storing Intermediary Velocity Field

In order to perform Fluid-Implicit Particle Advection update we need to store intermediary velocities that will be used at the end of the time-step. Those velocities are the ones that were transferred from the particles to the mesh; however, they are not free of divergence, since no pressure projection was performed. The intermediary velocity field is stored at node-locations, since it simplifies the interpolation procedure for cut-cells.

3.4.4 Transfer to Particles

PIC velocity update interpolates a new velocity from the grid to particles using a bilinear/trilinear interpolant. However, this update introduces dissipation, and interesting features of the flow quickly dampen. To alleviate that, the FLIP velocity update better enforces momentum conservation using a Lagrangian view of the flow: each particle, individually, should retain the same original velocity as time advances; if this is the done for all the particles inside the domain, momentum is exactly conserved. In the real world, particles also collide and interact with each other (BRACKBILL; RUPPEL, 1986), and this changes their velocity as time advances. These collisions and interactions are modelled in the FLIP scheme by interpolating the difference of the intermediate velocity field with respect to the mass-conserving velocity field. This is expressed as:

$$\mathbf{u}_p = \mathbf{u}_p + (\mathbf{u}_{\text{interp}} - \mathbf{u}^*_{\text{interp}}), \tag{3.14}$$

where $\mathbf{u}_p$ the particle's velocity, $\mathbf{u}^*_{\text{interp}}$ and $\mathbf{u}_{\text{interp}}$ are the interpolated velocities from the grid before and after projection, respectively.

Interpolation of data from the simulation mesh back to the particles is performed using our Spherical Barycentric interpolation scheme on cut-cells, which will be presented in Section 4.3; for regular cells, standard bilinear/trilinear interpolants are used. However, we found that using FLIP in cut-cells leads to instability, particularly for small cells; we conjecture that the inherent instabilities in the FLIP scheme are exacerbated in the presence of such small cells. Therefore, *we revert to a pure PIC approach* in cut-cells, using FLIP only in the broader domain.

3.4.5 Discussion

Our modified FLIP advection scheme allows us to construct a cut-cell-aware particle-based advection scheme that addresses an issue faced by previous Eulerian and semi-Lagrangian schemes in the presence of moving boundaries. Specifically, solid boundaries that sweep over the stationary Eulerian grid leave behind cells lacking velocity data. Consider the example in Figure 3.14: a volumetric circular solid (green) translates past to reveal the formerly inactive central cell (light blue). It suddenly becomes active again and its faces must be assigned valid velocity data before continuing the simulation. Mittal and Iaccarino (MITTAL; IACCARINO, 2005) dub these “freshly-cleared" cells. While physically, fluid velocities would simply advect along with the object and fill in the missing data, this is not true in the discrete case for Eulerian and semi-Lagrangian schemes.

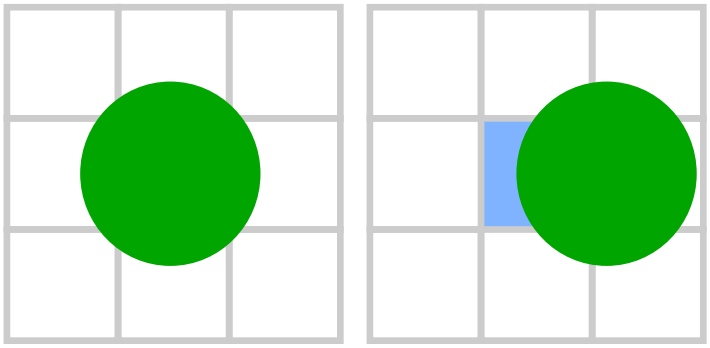


Figure 3.14: A volumetric object translating left-to-right reveals the uninitialized center cell over a time step.

For large volumetric solids, semi-Lagrangian advection suffices, since the velocities previously extrapolated into the solid are often reasonable. However, for relatively thin boundaries, (sub-)cell regions often go from one side of a moving solid boundary to another *in a single step*, and there is no extrapolated velocity already present “inside" the object. Using semi-Lagrangian advection, the end-of-trajectory velocity value which one would ordinarily use to begin backtracing *from* lies on the *wrong side of the boundary*, where the fluid may be flowing in entirely the wrong direction.

Guendelman et al. (GUENDELMAN et al., 2005) observe this issue, and instead reinitialize these cells by using an iterative averaging procedure to extrapolate data from nearby fluid cells on the same side which were not swept over, and hence contain valid data. However, this choice has two shortcomings illustrated in (Figure 3.15): in the case of closed invalid regions,

data must be created from scratch, or it may result in data being extrapolated quite long distances (e.g., in narrow regions).

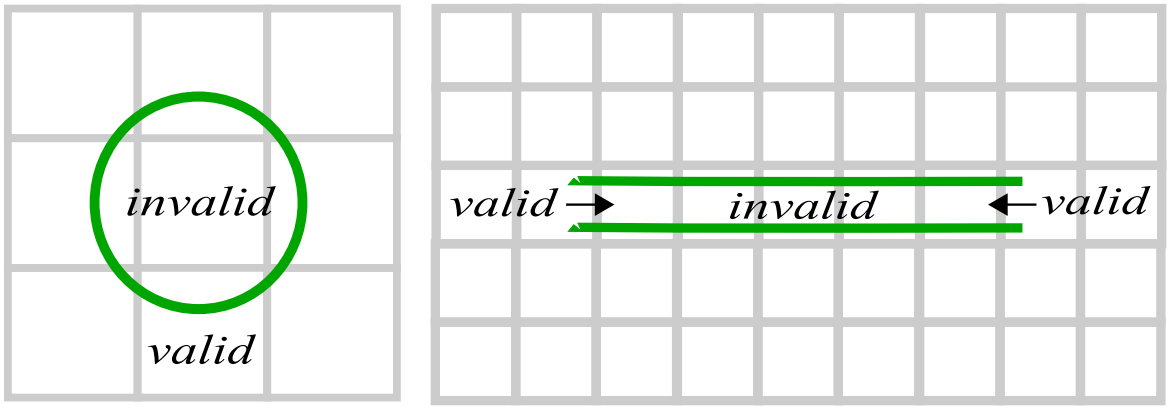


Figure 3.15: Failure cases for standard velocity extrapolation from valid into invalid (uninitialized) cells. Left: Closed regions cannot be extrapolated into. Right: Long narrow regions may require extrapolation across arbitrary distances.

We observe that with an appropriate PIC/FLIP implementation, this issue does not arise. Given a sufficient sampling of Lagrangian particles, there is no need for backtracing or extrapolation; velocity data carried by the particles travels *forwards* in tandem with the solid boundary to fill the freshly cleared cells, much like in the physical world. Semi-Lagrangian methods create particles on velocity locations on-the-fly and update those particles *back in time* in order to find new advected velocities; thus, a moving object will invalidate cells swept over by it, since semi-Lagrangian particles often travel in the opposite direction of moving boundaries. We maintain a good sampling of particles throughout by placing a user-defined lower and upper bound on the number of particles *per unit area*, and reseeding on each *sub-cell* independently. To complete our advection scheme, we just need mechanisms to transfer particle velocity data to and from the cut-cell mesh.

3.5 Summary

This chapter discussed our proposed cut-cell method for handling thin boundaries, narrow gaps and irregular shapes. We introduced boundary cut-cell meshes and briefly discussed how they are created - more details about cut-cell construction will be presented in the next chapter. Section 3.2 presented our topological pressure projection method. The proposed pressure projection algorithm is able to converge with second-order accuracy in space for pressure samples. Moreover, a feasible approach for handling cells with degenerate geometry was

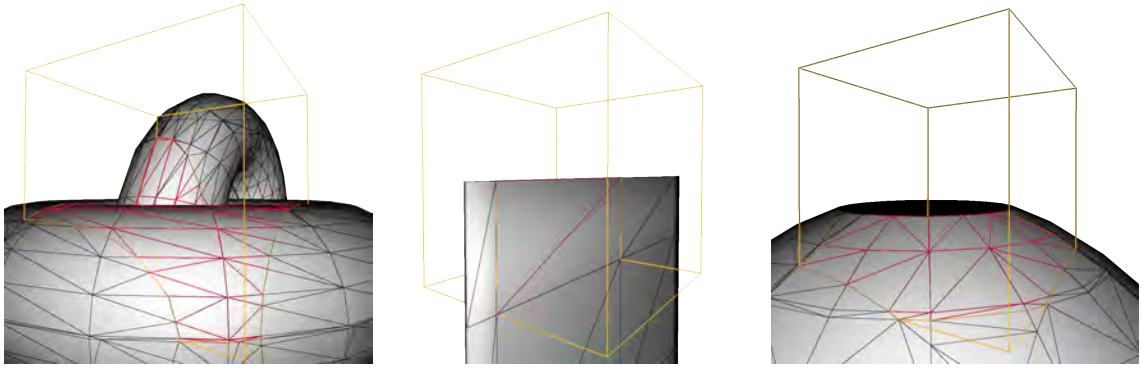
proposed. Section 3.3 discussed details about the interpolation and reconstruction of velocities on cut-cells. We showed that our boundary-conforming velocity interpolant is able to produce streamlines that do not interpenetrate the object's mesh, while standard one-sided bilinear interpolants do not guarantee this property. We reconstructed free-slip and no-slip velocities for cut-cells using a weighted least squares approach coupled with propagation. Finally, Section 3.4 presented our FLIP/PIC advection scheme, detailing its steps and highlighting the differences with the traditional approach. The next chapter presents specific implementation details our method.

4 IMPLEMENTATION

This chapter discusses specific implementation details of the proposed algorithm. Section 4.1 presents details about the cut-cell generation scheme and Section 4.2 discusses our free-slip velocity reconstruction using the weighted least squares method. Our prototype was implemented using C++, CUDA and GLSL.

4.1 Cut-cell splitting method

Cut-cell generation is not a trivial task, especially when considering that an arbitrary number of complex 3-D objects may subdivide a regular voxel multiple times. Besides that, we allow the geometries of the objects to have holes and border edges, which also complicate cut-cell generation. The algorithm presented in this thesis is able to successfully handle these cases, as shown in Figure 4.1.



(a) (b) (c)

Figure 4.1: Cut-cell examples: (a) cut-cell of intersected tori meshes; (b) cut-cell example for a mesh with border edges; (c) cut-cell of a mesh with holes.

Initially, our cut-cell generation algorithm constructs a single non-manifold mesh for each regular voxel that is in contact with any geometric element on the domain. For voxels that encapsulate mesh regions with no holes or border edges, each non-manifold mesh is then split into two or more manifold parts, each of which generates an independent cut-cell. For voxels containing meshes with holes or border edges, we allow non-manifold cut-cells that will receive special treatment in later phases of the simulation. However, a voxel could have a combination of the above cases, i.e., at least one mesh without holes and at least one mesh with holes sharing the same voxel. Our splitting algorithm is able successfully identify non-manifold parts that cannot be split.

Although splitting non-manifold geometry into separate manifold parts is a well-known and studied problem, most of the methods proposed so far focus on geometry noise removal or fixing defective computer aided design (CAD) models (GUÉZIEC et al., 2001). Moreover, these algorithms often require user interaction to resolve ambiguity among multiple possible solutions. Our algorithm, on other hand, does not require user-interaction to construct cut-cells on difficult scenarios. Algorithm 3 provides an overview of our cut-cell generation method.

Algorithm 3 Cut-Cell Generation

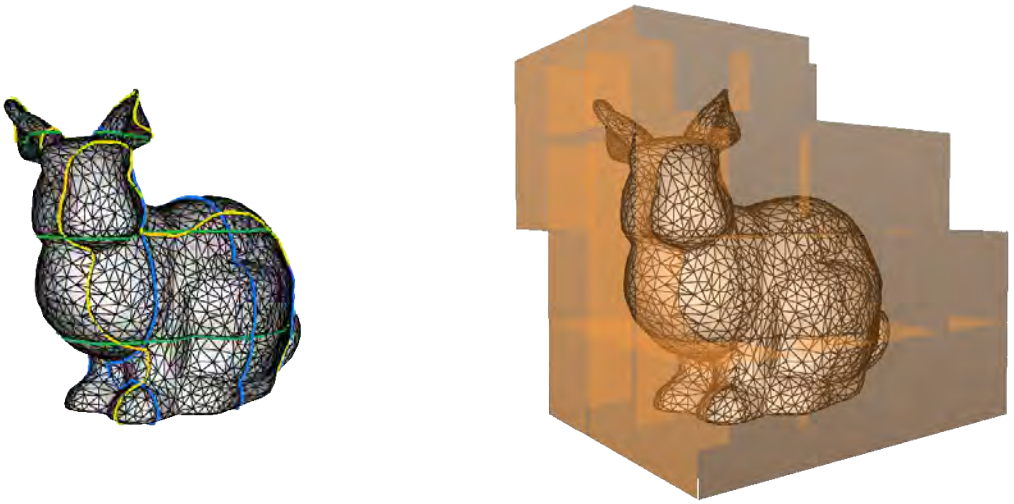
```
for each mesh in domain do
    Compute intersections between the grid and the mesh (Section 4.1.1)
    Compute 2-D cut-faces from obtained intersections (Section 4.1.2)
    Generate a list of boundary 3-D voxels (Section 4.1.3)
    Initialize non-manifold meshes (Section 4.1.4)
    while Non-visited faces do
        Depth-first splitting algorithm (Section 4.1.5)
    end while
end for
```

4.1.1 Computing intersections between the grid and meshes

Firstly, we compute all intersections between regular grid faces and the meshes representing the objects. In 2-D, this reduces to computing crossings between grid edges and line segments; the output is a set of ordered crossing points, classified by the type of the grid edge that they cross (vertical or horizontal). These crossings points are added to the original line that represents the object, ensuring that each line segment spans only a single cut-cell. In 3-D, the output of computing intersections with grid faces are three different sets of lines for each mesh (Figure 4.6). Each set represents one of the different plane orientations (vertical, horizontal, transversal) that constitute 3-D regular grids. The output of finding intersections for the Bunny mesh is shown in Figure 4.2a, and these are computed using the Computational Geometry Algorithms Library (CGAL).

Analogous to the 2-D case, the computed 3-D intersections are added to the original mesh. This step ensures that triangles of the mesh does not span different regular grid voxels, facilitating the non-manifold mesh subdivision process. Adding sets of connected lines can modify the original mesh topology and generate quadrilateral mesh patches. Even though our cut-cell assembly or interpolation algorithms do not need a triangular mesh as input, we use CGAL's

library (CGAL, 2016) to triangulate the modified geometry mesh to speed up point-in-polygon tests that are needed in the advection stage.



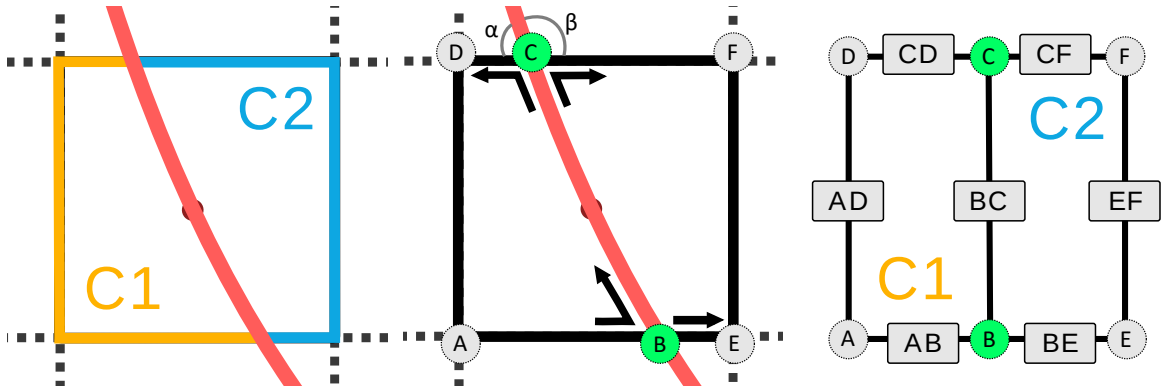
(a) (b)

Figure 4.2: Intersection computation with the Stanford bunny and generation of boundary voxels: (a) lines resulting from the intersection of the grid planes and the object's mesh. Blue lines represent intersections with X-aligned planes; green lines correspond to intersections with Y-aligned planes; and yellow lines correspond to intersections with Z-aligned planes; (b) candidate cells tagged for non-manifold mesh construction and subdivision.

4.1.2 Cut-face computation in 2-D

After computing the intersections between grid planes and object meshes, we can compute 2-D cut-faces in 3-D. Each computed cut-face will belong to a grid plane orientation (vertical, horizontal, transversal). The algorithm in 2-D follows the same logic as its 3-D counterpart, except that it is in a lower dimension: 2-D cut-faces are replaced by the computation of 1-D cut-edges; a list of boundary 3-D voxels is replaced by a list of boundary 2-D cells; non-manifold meshes in 2-D are initialized and, lastly, a depth-first algorithm will be performed until all the cut-edges are visited. The first step is not needed, since the intersections between grid planes and the mesh are sets of 2-D lines, so they are just incorporated into the 2-D cut-cell generation algorithm. Also, computing 1-D cut-edges follows logically from computing intersections, since they are created from partial edge fractions from the crossing of geometry 2-D lines and grid edges.

We will detail our 2-D depth-first splitting algorithm, which works similarly for 3-D, by illustrating its steps. Figure 4.3a shows a 2-D regular cell subdivided by some object geometry into two cut-cells C1 and C2. The non-manifold mesh has T-junction nodes represented by the green vertices B and C in Figure 4.3b. Using this representation, a graph that connects edges and vertices according to their adjacency information is constructed, as shown in Figure 4.3c. In order to split the graph into cells C1 and C2, we perform a depth-first search, starting on an arbitrary grid edge. Once on an edge, the algorithm always accesses a node to its right (or to its left), following a counter-clockwise (or clockwise) order that must be respected until the end of the algorithm.



(a) (b) (c)

Figure 4.3: Schematic view of the non-manifold splitting algorithm in 2-D: (a) desired solution of the splitting algorithm showing cells C1 and C2; (b) non-manifold mesh structure, green vertices are T connections, in which the degree of the connectivity graph is greater than 2. The arrows point the direction that the algorithm may choose on vertices B and C. (c) A schematic view of the non-manifold graph structure, which connects both edges and vertices.

The non-manifold splitting algorithm follows from edges to vertices and vice-versa using a pre-defined orientation when the degree of the nodes is equal to 2. However on T-junction nodes (vertices B and C, degree > 2) the algorithm has to choose which path it should follow to create a consistent manifold mesh. The key constraint is that all edges of the split manifold must belong to the same side of the geometry boundary. Alternatively we could use a greedy approach coupled with some backtracking mechanism, which follows all possible paths and rolls back to the last valid configuration when it fails. We adopt a simpler and more efficient solution that uses location and geometry information to correctly choose which path the algorithm should take.

Given the example shown in Figure 4.3c, the algorithm starts with an initial non-visited edge, and it navigates through the graph accessing the vertices and checking which edges are connected to it, tagging each edge that it accesses as visited. Let AD be the initial edge, and assume that the algorithm proceeds in a counter-clockwise order. Then it accesses vertex A,

which has only 2 neighbors (one of them is the edge AD that was accessed last), so it simply follows to the next non-visited edge AB.

The AB edge is classified as a *grid edge*, since it is on top of an edge of the regular grid; an edge could also be classified as a *geometry edge*, when it lies on top of a geometry edge (e.g., BC). The algorithm continues accessing the next non-visited vertex, which is B, checking its connectivity. In this case, it has to choose between the edges BC and BE (bottom dark arrows in Figure 4.3b). Here, it uses its *location information* to choose correctly: when the algorithm is currently evaluating a grid edge (AB) and has to choose between a geometry edge (BC) and another grid edge (BE), it will always choose the geometry edge. This approach is sufficient to build a consistent non-manifold cut-cell mesh which does not contain edges from different sides of the obstacle's geometry.

The algorithm continues to the next non-visited vertex, which is vertex C. In this case it also has to choose between two different edges (CD and CF), but those edges are both grid edges. Here, the algorithm uses *geometry information*: we calculate the *relative signed angle* between the vector that is arriving on the vertex C (BC), and the vectors that have its origin on vertex C (CD and CF). These angles, are depicted as α and β in Figure 4.3b. We choose the smaller angle between those vectors to continue, which in this case is the edge CD, since α is the smallest angle. This is equivalent to a "left-turn" and it follows because we chose a counter-clockwise orientation. If one chooses a clockwise orientation we must choose the greater angle, which is equivalent to a "right-turn". The algorithm will end when it reaches the initial edge AD, since now all possible paths from this edge have already been visited.

We notice that our solution works for graphs of arbitrary degrees; however all of our simulation scenarios had connectivity equal to or smaller than 3 (Figure 4.5). When the connectivity is higher than 3 we have degenerate cases, where mesh points are lying exactly on top of grid faces, edges or nodes, complicating other parts (such as triangulation for point-in-cut-cell tests) of the simulation pipeline.

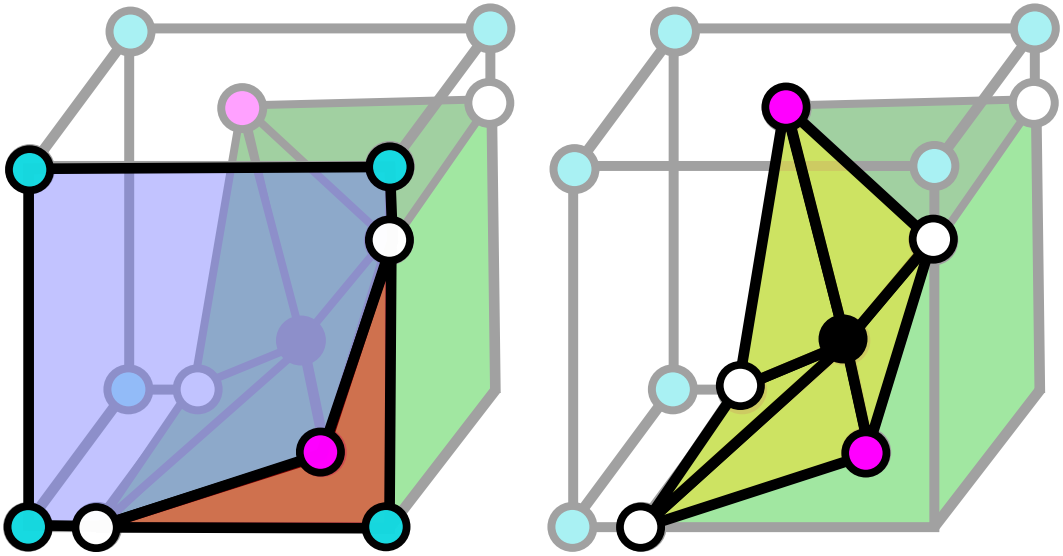
4.1.3 Boundary voxel list generation

The next step is to identify grid voxels that are going to be assembled into non-manifold meshes (Figure 4.2b). This is accomplished by tagging all voxels containing our 2-D cut-faces discussed in previous section. To quickly do that, we compute the regular-grid index locations of each 2-D cut-face by dividing its centroid location by the regular-grid spacing. This will give us the absolute index relative to the dimension of the grid of each 2-D cut-face. Then, depending

on the plane that the cut-face belongs to, we tag the 2 voxels that share that face. For example, let a regular-grid index be expressed as (i, j, k), where i, j, and k correspond to the X, Y, and Z coordinates, respectively. For a cut-face that belongs to the XZ plane, the voxels that are going to be tagged are the ones with (i, j, k) and $(i, j + 1, k)$ indices. The output of the tagging procedure is the list of non-manifold meshes that are going to be (possibly) decomposed into manifold parts.

4.1.4 Construction of non-manifold meshes

With the list produced by the tagging procedure, we can construct non-manifold meshes for each cut-cell. To accomplish this, the algorithm has to find and categorize all faces inside the same regular-grid voxel. Similar to 2-D cut-faces computation, there are two categories of faces: *grid faces* (Figure 4.4a) and *geometry faces* (Figure 4.4b). Grid faces are polygonal, most of the time having more than 3 segments, and are contained inside a plane that constitutes the original regular grid. These faces are computed by the 2-D version of the non-manifold algorithm and are always shared by only 2 distinct cut-cells. Geometry faces are those that belong to the original mesh, will always be triangular and span only one regular voxel.

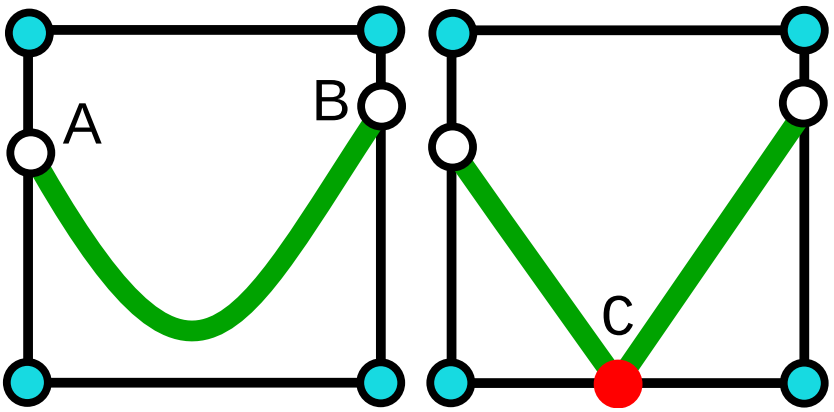


(a) (b)

Figure 4.4: Different cut-cell face types for 3-D: (a) grid faces (highlighted in blue and red) are generated by the 2-D non-manifold splitting algorithm and can be arbitrary planar polygons; (b) geometry faces (highlighted in yellow) belong to the original mesh and are always triangular.

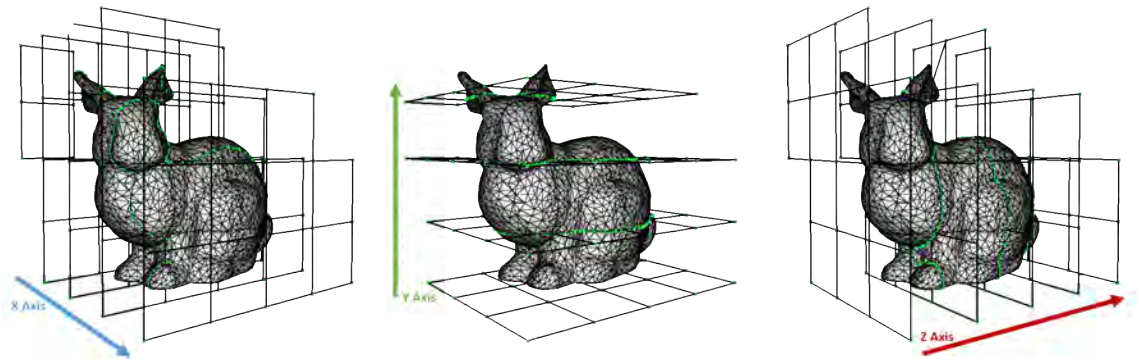
Faces from the non-manifold mesh are initialized according to their respective types. Grid faces are categorized by their orientation (XY, XZ or YZ planes) by checking the corresponding

face normal; then, these faces are added to a 3-D matrix that will encode their index location within the regular grid. For geometry faces, only a mapping to the regular-grid location is needed. The location of both geometry and grid faces in regular-grid index coordinates is calculated by computing the centroid of the face and dividing it by the grid spacing. Lastly, all faces grouped together in a regular-grid voxel are connected by a graph, whose adjacency map is built using faces' edges information. Non-manifold meshes will have nodes with degree higher than 2 (T-junctions). T-junction nodes will be the key elements on the splitting method that generates separate manifold parts.



(a) (b)

Figure 4.5: Node degrees examples: (a) two nodes (A and B) with degree equal to 3 are connected to grid edges that are at opposite sides of the mesh boundary (green); (b) when a node of the object lies exactly on top of a grid edge (red C node) it creates a T-junction with degree > 3, since now 4 edges (two grid edges and two geometry edges) are connected to it.



(a) (b) (c)

Figure 4.6: (a) 2-D horizontal cell faces (X aligned) initialized by the 2-D non-manifold algorithm using slices intersection information; (b) 2-D vertical cell faces (Y aligned); and (c) 2-D transversal cell faces (Z aligned).

4.1.5 Non-Manifold mesh splitting in 3-D

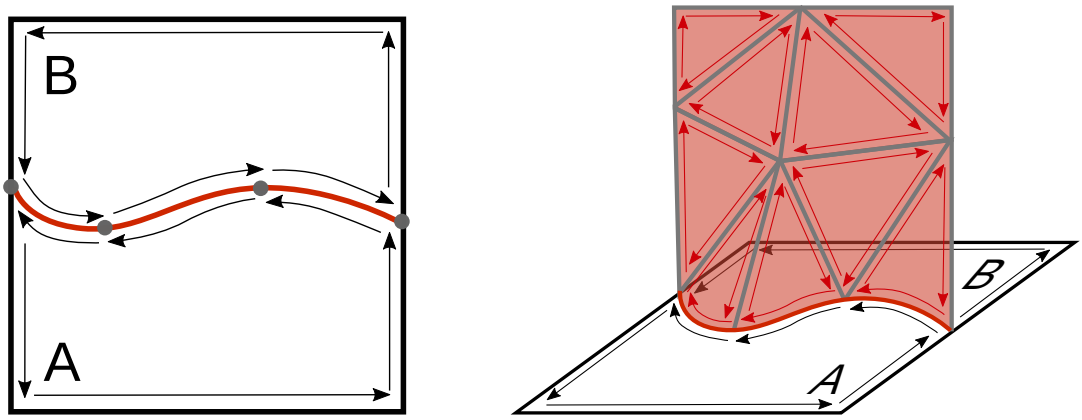
The non-manifold splitting in 3-D is analogous to the 2-D version. The graph building procedure is similar, with faces substituting edges and edges substituting vertices. We can keep the same strategy using location information when choosing the path to go from a grid face (analogous to grid edge) to a geometry face (analogous to geometry edge). However, since each face now has at least 3 edges, it is not possible to define an access rule that follows a counter-clockwise (or clockwise) order through the entire non-manifold geometry. This means that the 2-D strategy of using relative signed angle cannot be applied for the 3-D case.

A key property for cut-cells that can be used for path selection is that all its faces must always be on the same side (interior or exterior) of the geometry mesh. Inside geometry tests for grid faces are often costly and non-robust for 3-D, but they can be completely avoided by using the winding order defined by the 2-D algorithm. All 2-D faces from a non-manifold mesh respect the same winding order (clockwise or counter-clockwise). Each geometry edge (red edges on Figure 4.7a) is composed by half-edges (directed edges, arrows on Figure 4.7a) with opposite directions.

To verify if a *grid face* is interior or exterior to the mesh, we devise a simple but effective solution. The *half-edges* of an edge shared by a grid-face and a geometry face are compared: if their orientations match (Figure 4.7b), it means that the grid face is exterior the mesh; otherwise, the grid face is interior to the mesh. This is only possible because the half-edges of our mesh patches have their orientation defined consistently to the mesh original normal and its containing regular-grid voxel.

Therefore, when the algorithm transitions from a grid face to a geometry face using location information, we verify if the grid face is interior or exterior to the mesh and store this relative position. A transition from a grid face to a geometry face will always happen before a transition from a geometry face to a grid face, because the algorithm always starts in a grid face. Whenever the decision between two different grid faces has to be made, the algorithm simply follows the same order stored on the transition of a grid face to a geometry face. For the example shown in Figure 4.7b, the geometry half-edges (black arrows) of the foremost grid face match their orientation with the half-edges (red arrows) of geometry faces (red triangles), which means that this grid face is exterior to the mesh. With this simple rule, we are able to decide correctly between paths in T-junctions, and no backtracking is needed.

Some cut-cells examples generated by the proposed algorithm can be seen on Figure 4.8.



(a) (b)

Figure 4.7: Half-edges orientations for faces "A" and "B": (a) planar 2-D view of cut-faces. Red edges represent the interface between the two grid faces and the objects geometry. Each of those edges on the interface are represented on grid faces by opposing half-edges. In (b), schematic 3-D visualization of the polygon winding orientation of same set of geometry and grid faces. Geometry half-edge orientations of grid face "B" are omitted for visualization purposes.

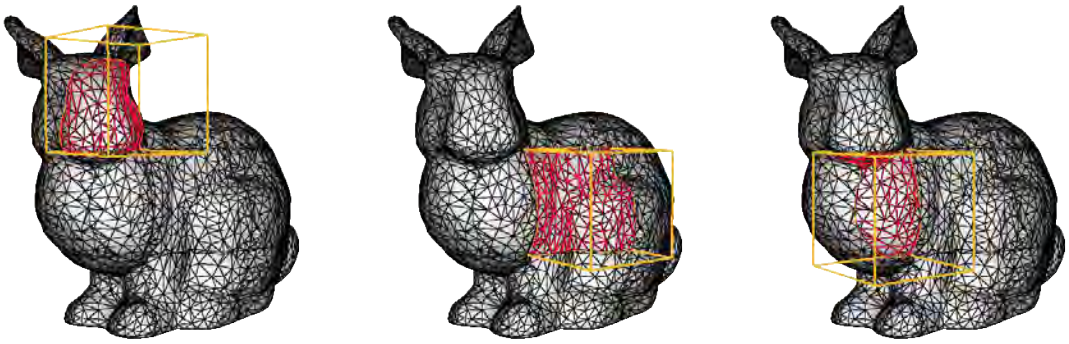
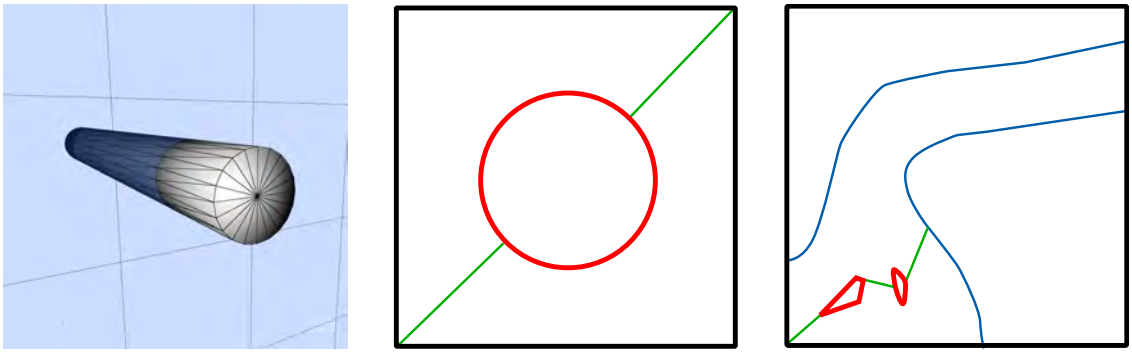


Figure 4.8: Examples of cut-cells generated with our algorithm for the Stanford Bunny.

4.1.5.1 Disconnected regions

In 3-D, there may be cases in which objects cut a hole through a single grid face (e.g., a long cylinder as in Figure 4.9). In this case, the intersection line between the plane and the geometry is completely contained inside a grid face, and the non-manifold splitting algorithm in 2-D is oblivious to interior disconnected regions. Therefore, disconnected regions must be connected to outer grid face edges in order to provide valid 2-D faces for the non-manifold splitting algorithm (center and right images of Figure 4.9).

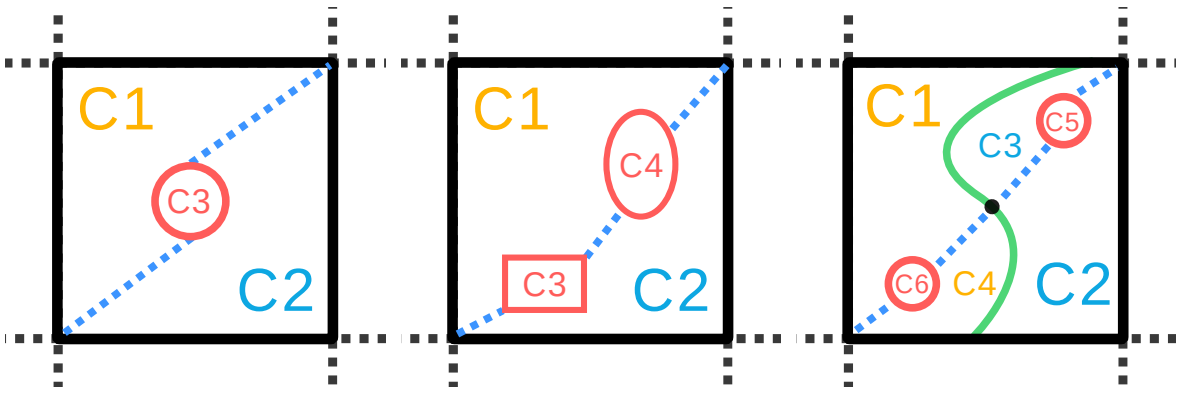
We devise a simple algorithm to connect disconnected regions to outer grid face edges. It starts by ordering the centroids of disconnected regions relative to one axis in a list (e.g. ordering by the Y axis). Then it sequentially accesses the vector, connecting each subsequent pair of disconnected regions by their closest points (Figure 4.10a). If this connection between disconnected regions crosses through another geometry edge (Figure 4.10c), the algorithm simply



(a) (b) (c)

Figure 4.9: Disconnected regions examples: (a) a cylinder cuts through a single regular grid face; (b) cylinder intersection (red edge) is completely contained inside a grid face, not crossing any grid edges. So, the algorithm adds two connection edges (green edges) from the disconnected region to regular grid nodes generate valid cut-faces In (c), a more complex example with multiple disconnected regions (red edges) connected to geometry intersections (blue edges).

adds additional connections to the crossed geometry edge. Lastly, it connects the first (lower relative to the order of the centroids) disconnected region to the closest outer regular-grid face vertex; a similar procedure is done for the last (upper relative to the order of the centroids) disconnected region as well. We notice that this algorithm might fail, if the disconnected regions are non-monotonic or be disposed in a intricate fashion. However, for all tested configurations, the algorithm performed correctly; one example of a complex case successfully handled by our algorithm can be seen on Figure 4.9 (c).



(a) (b) (c)

Figure 4.10: Schematic view of different disconnected regions. (a) A single circular hole (C3) through a grid face. (b) Two disconnected regions (C3 and C4) connected by their closest points. (c) Closest points between disconnected regions (C5 and C6) obstructed by another geometry face. In this case the algorithm adds connections directly to the geometry face between disconnected regions and new regions are generated by those connections (C3 and C4).

4.2 Least squares velocity fitting

The method of least squares is a standard approach to determine approximation solutions of overdetermined systems by minimizing the sum of the squares of the errors made in the results of every single equation. Consider the following overdetermined system, for an m by n matrix N, where $m > n$:

$$N\mathbf{u} = \mathbf{z} \tag{4.1}$$

We can multiply the whole equation by N^T, which will yield a n by n matrix:

$$N^T N\mathbf{u} = N^T\mathbf{z} \tag{4.2}$$

The result of $N^T N$ now can be inverted, since the matrix is square. The product $N^T N$ is also positive definite. To demonstrate that, assume $\mathbf{x} \neq 0$, then

$$\mathbf{x}^T(N^T N)\mathbf{x} = (N\mathbf{x})^T(N\mathbf{x}) = ||N\mathbf{x}||_2^2 > 0. \tag{4.3}$$

Multiplying Equation (4.2) by the inverse of $N^T N$ will yield the standard solution for linear squares systems:

$$(N^T N)^{-1}(N^T N)\mathbf{u} = (N^T N)^{-1}N^T\mathbf{z} \tag{4.4}$$

We can define the pseudo-inverse of N as $N^+ = (N^T N)^{-1}N^T$; thus

$$I\mathbf{u} = N^+\mathbf{z} \tag{4.5}$$

4.2.1 Weighted least squares

The weighted least squares approach can be written in the following form

$$WN\mathbf{u} = W\mathbf{z}, \tag{4.6}$$

where the W matrix is a m by m diagonal matrix of the weights from each sample contribution. The product WN is a m by n matrix, which cannot be inverted. To fix that, one can multiply the equation by N^T:

$$N^T WN\mathbf{u} = N^T W\mathbf{z}. \tag{4.7}$$

The product $N^T W N$ is a n by n matrix that can be inverted. One can multiply the both sides of the equation by the inverted expression:

$$(N^T W N)^{-1}(N^T W N)\mathbf{u} = (N^T W N)^{-1} N^T W \mathbf{z}. \tag{4.8}$$

Then, to find $\mathbf{u}$, we have to just solve the right side of the equation above. Our weights are obtained as

$$W_{ii} = \frac{1}{r_{ii} + \epsilon}, \tag{4.9}$$

where r_{ii} are the distances between the mixed node i and the location of its velocity node.

4.3 Spherical Barycentric Coordinates

We adopted the Spherical Barycentric Coordinates (SBC) interpolant in our 3-D implementation. This interpolant was chosen over Mean Value Coordinates (MVC) since SBC is able to work on polyhedra with arbitrary polygonal meshes, while MVC only works for polyhedra with triangular faces. Our cut-cells are triangulated for point-in-polygon tests; however, the triangulation process introduces skew triangles that may hinder the overall accuracy of the interpolation (for some examples, check (LANGER; BELYAEV; SEIDEL, 2006)). A cut-cell triangulation example can be seen on Figure 4.11. Notice how the triangulation process introduces small skew triangles.

Spherical Barycentric Coordinates describe the position of a point inside a polyhedra with respect to all of its spherical polygons. A spherical polygon is found by projecting a polygon into a sphere of unit radius. For interpolating a discrete function defined at the boundaries of a irregular polyhedra M to a point $\mathbf{p}$ inside M, we want to find a set of weights λ_i such as: s

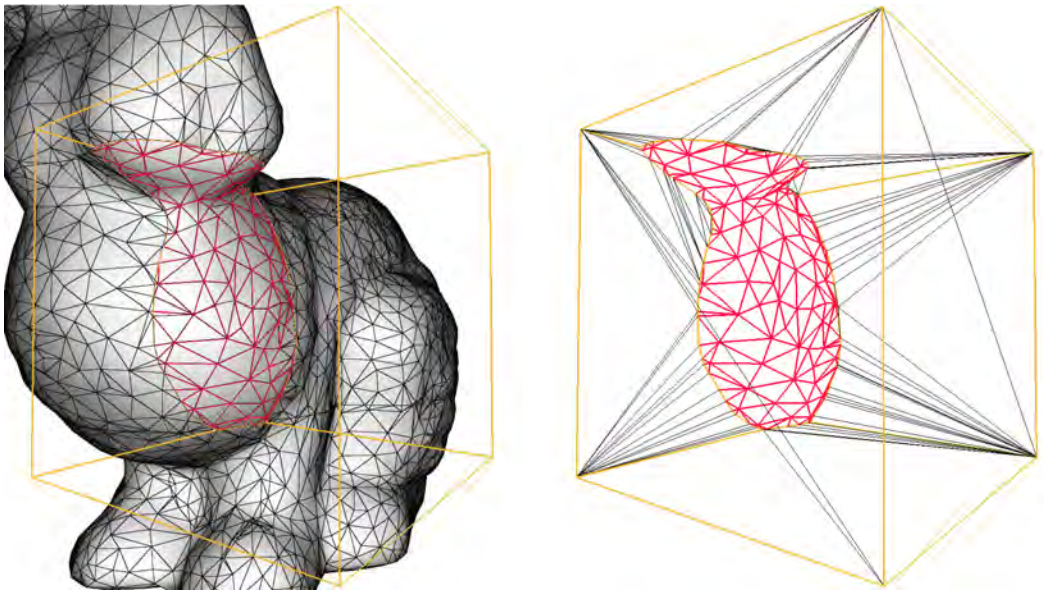
$$\mathbf{p} = \sum_{i}^{n} \lambda_i \mathbf{x}_i, \tag{4.10}$$

where $\mathbf{x}_i$ are the vertices of the polyhedra. Although SBC does not satisfy the partition of unity (i.e., $\sum_i^n \lambda_i = 1$) and may fail reproducing constant functions over a polyhedra, it satisfies the linear precision property:

$$\sum_{i}^{n} \lambda_i f(\mathbf{x}_i) = f(\mathbf{p}), \tag{4.11}$$

where f is an arbitrary function over the domain spanning the polyhedra. The linear precision property is important for defining velocity field streamlines that do not interpenetrate the object

mesh. As an arbitrary interpolation point approaches a given face, the linear precision property guarantees that the interpolated value will converge to a linear combination of the values defined at the vertices of that face. This property does not hold in radial basis interpolants, for example.



(a) (b)

Figure 4.11: Triangulation of a cut-cell: (a) original bunny cut-cell polyhedron with arbitrary polygonal faces; (b) triangulated cut-cell.

Langer et al. (LANGER; BELYAEV; SEIDEL, 2006) do not explicitly present an algorithm and a straightforward SBC implementation is challenging. Therefore, in Algorithm 4, we present the steps of the Spherical Barycentric Coordinates algorithm employed in our implementation. The algorithm starts by finding v_i values for each face on the polyhedra and normalizing the corresponding face vector. Then for each point in each face, it calculates a *distinct* λ_{ij}^* value and this means that for a given unique vertex several instances of λ_{ij}^* will be associated to it. Therefore, all values of λ_{ij}^* are summed to λ_i, which relates each weight to a unique vertex in the polyhedra. Finally, all the weights λ_{ij}^* are normalized by their sum λ_i, being ready for interpolating values for points inside the polyhedra. Some velocity interpolation examples are shown below in Figure 4.12.

Algorithm 4 Spherical Barycentric Coordinates Interpolation

for each polygon with index i in mesh **do**
  for each point with index j in polygon **do**
    //Auxiliary vectors calculation
    $\mathbf{a}_j = \mathbf{x}_j - \mathbf{p}$
    $\mathbf{a}_{j+1} = \mathbf{x}_{j+1} - \mathbf{p}$
    $\mathbf{n}_i = \mathbf{a}_j \times \mathbf{a}_{j+1}$
    $\mathbf{n}_i = \frac{\mathbf{n}_i}{||\mathbf{n}_i||}$
    //Auxiliary angle calculation
    $\theta = \arccos(\frac{\mathbf{a}_j \cdot \mathbf{a}_{ji+1}}{||\mathbf{a}_j|| ||\mathbf{a}_{j+1}||})$
    $\mathbf{v}_i = \mathbf{v}_i + \mathbf{n}_i * 0.5 * \theta$
  end for
  //Normalize $\mathbf{v}_i$
  $\mathbf{v}_i = \frac{\mathbf{v}_i}{||\mathbf{v}_i||}$
  for each point with index j in polygon **do**
    //Auxiliary vectors calculation
    $\mathbf{a}_{j-1} = \mathbf{v}_i \times \left(\frac{\mathbf{x}_{j-1} - \mathbf{p}}{||\mathbf{x}_{j-1} - \mathbf{p}||}\right)$
    $\mathbf{a}_j = \mathbf{v}_i \times \left(\frac{\mathbf{x}_j - \mathbf{p}}{||\mathbf{x}_j - \mathbf{p}||}\right)$
    $\mathbf{a}_{j+1} = \mathbf{v}_i \times \left(\frac{\mathbf{x}_{j+1} - \mathbf{p}}{||\mathbf{x}_{j+1} - \mathbf{p}||}\right)$
    //Auxiliary angles calculation
    $\alpha_{j-1} = \arccos(\mathbf{a}_{j-1} \cdot \mathbf{a}_j)$
    $\alpha_j = \arccos(\mathbf{a}_j \cdot \mathbf{a}_{j+1})$
    $\theta = \arccos(\mathbf{v}_i \cdot \mathbf{a}_j)$
    //Lambda and denominator calculation
    $\lambda^*_{ij} = \frac{\tan(\alpha_{j-1} * 0.5) + \tan(\alpha_j * 0.5)}{\sin(\theta)}$
    $d = d + (\tan(\alpha_{j-1} * 0.5) + \tan(\alpha_j * 0.5)) \tan(\frac{\pi}{2} - \theta)$
  end for
  for each point with index j in polygon **do**
    $\lambda^*_{ij} = \frac{\lambda^*_{ij}}{d}$
  end for
end for
//For all faces connected to a given vertex, sum each λ^*_{ij} *to the vertex*
$\lambda_j = \sum_i^n \lambda^*_{ij}$
//Calculate the sum of all weights and normalize
$s_\lambda = \sum_j^n \lambda_j$
for each vertex with index i in vertices **do**
  $\lambda_i = \frac{\lambda_i}{s_\lambda}$
end for

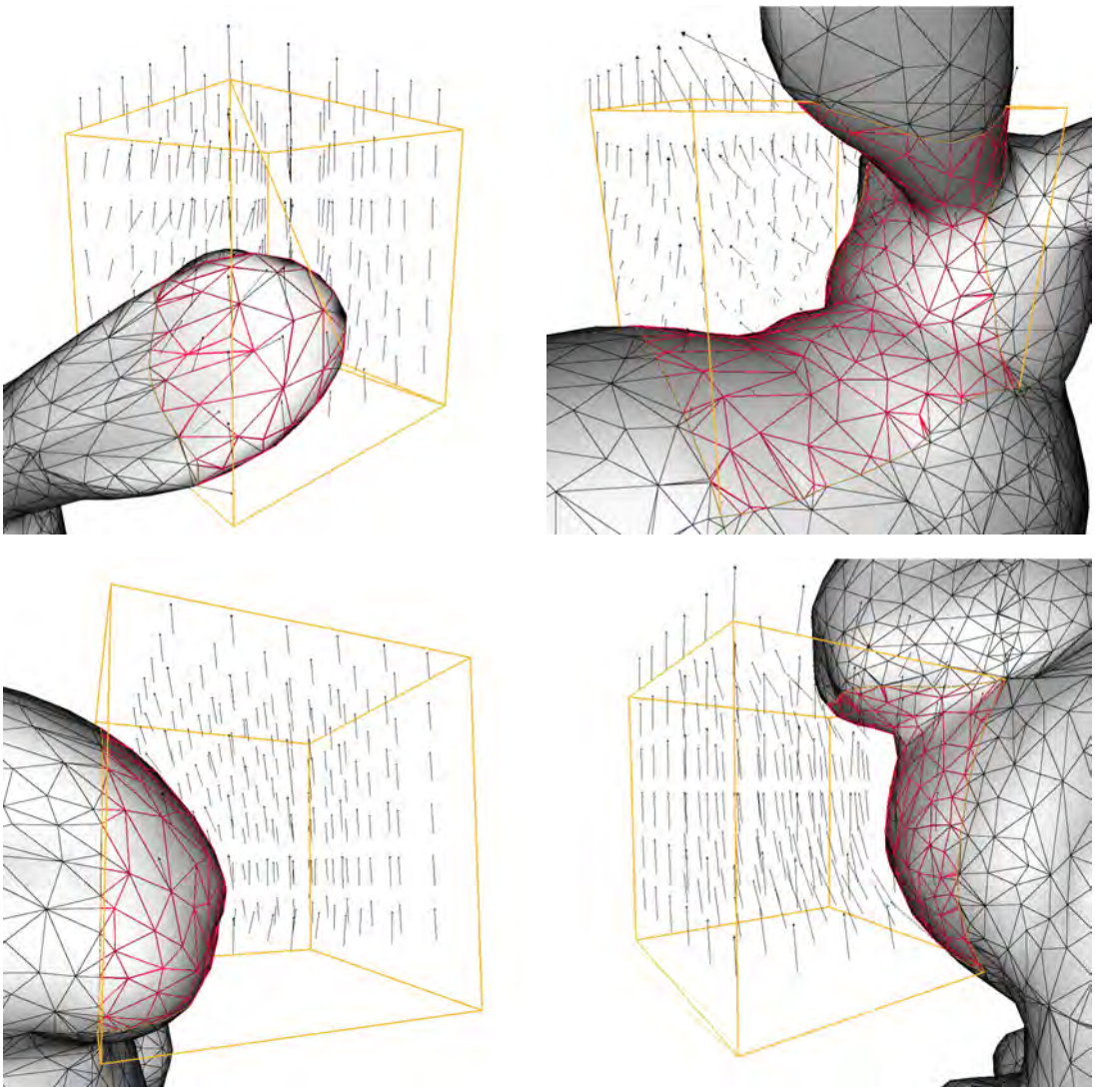


Figure 4.12: Velocity interpolation examples using Spherical Barycentric Coordinates on cut-cells.

5 RESULTS

This chapter discusses the results obtained applying our cut-cell algorithm to various challenging regular-grid scenarios involving thin objects, irregular shapes, and narrow regions. We present our prototype in 3-D, except for certain flow characteristics, which are better demonstrated in 2-D. Before discussing our results, we firstly present timings and settings (Table 5.1); all our results are computed using a single core of an Intel i7-2600 CPU at 3.4 GHz with 8GB memory. The robust intersection processing needed for mesh generation is handled with the CGAL library (CGAL, 2016). We simulated a single time-step per frame of the video.

While we focus primarily on coarse grids, Table 5.2 shows how our technique scales with increasing grid resolution. It provides some cut-cell statistics and performance numbers for a fluid flow around a Bunny model (5,002 triangles) with grid resolutions ranging from 8^3 up to 256^3. Timings were obtained by averaging five measurements with identical settings. As expected, as the number of regular cubic cells increases by a factor of 2^3 from one grid resolution to the next, the number of cut cells only increases by a factor of 2^2. Consequently, the *percentage* of cut cells (relative to the number of grid cells) reduces by a factor of 2; that is, for any given solid geometry, the *relative* overhead associated with handling irregular cells decreases as grid resolution increases. The cut-cell generation time is dominated by CGAL operations (ranging from 80% to 90% of the time for resolutions up to 128^3). The identification of all faces (both mesh and grid ones) incident to each mixed node is currently performed in a brute force and unoptimized fashion, and thus its cost (column *Mixed Nodes* in Table 5.2) increases by a factor of roughly 15 from one resolution to the next; this rapid growth ultimately causes it to reduce CGAL's relative contribution to the overall time at the 256^3 grid level. We expect that optimizing this procedure will significantly accelerate cut-cell generation at high resolutions. The rightmost columns of Table 5.2 show the advection and projection times involved in simulating a single time step.

Our simulations do not apply vorticity confinement or other artificial turbulence-creation mechanisms, although these could be easily added. Our 3-D examples use 64 or 128 PIC/FLIP particles per cell, while our 2-D examples use 32 or 64 particles; these counts are higher than normal because we want to ensure that even small or skinny cells are sufficiently well-sampled. A smart adaptive particle sampling scheme would likely bring these values down with minimal impact on the results.

The presentation of the results is organized as follows: Section 5.1 shows scenarios with infinitesimally thin objects, where the challenge consists of creating a representation that avoids

Example	Advection	Projection	Total	Meshing	Grid Dims	No. Cut-Cells
Linked Tori - FS	0.372	0.040	0.597	0.443 (once)	$7 \times 7 \times 7$	88
Bunny - NS	0.343	0.004	0.516	1.527 (once)	$7 \times 7 \times 7$	78
Bunny - FS	0.490	0.004	0.881	1.469 (once)	$7 \times 7 \times 7$	78
Dragon (f) - FS	1.124	0.301	2.686	7.984 (once)	$19 \times 14 \times 11$	1298
Dragon (c) - FS	0.880	0.172	1.986	4.446 (once)	$11 \times 9 \times 6$	326
Disk - FS	0.319	0.118	0.63	0.197	$25 \times 21 \times 13$	137
Disk - NS	0.296	0.105	0.554	0.200	$25 \times 21 \times 13$	137
Paddle - NS	0.290	0.148	0.597	0.153	$23 \times 10 \times 6$	67

Table 5.1: Timing and parameters for 3-D simulations: NS stands for no-slip and FS stands for free-slip boundary conditions. Timing information is in seconds per frame, and is computed as an average over the first 3-4 frames. Total time excludes meshing, listed separately. For moving geometry, the cut-cell count is given for the first frame, and the cut-cell meshing time is per frame. For static meshes, meshing occurs only once at the start, as stated on the table. All examples use 64 particles per cell, except the Linked Tori with 128. In static examples, cut-cells are generated only once; otherwise, meshing times are computed per frame.

Grid	# Cut-Cells	% Cut-Cells	# Polygons	CC (sec)	CGAL (sec)	CGAL (%)	MN (sec)	Advect (sec)	Project (sec)
8^3	90	17.58	7,810	0.82	0.68	82.06	0.0004	0.54	0.006
16^3	316	7.71	10,636	1.31	1.15	87.95	0.0012	0.71	0.045
32^3	1,359	4.15	17,502	2.81	2.55	90.43	0.0168	1.24	0.956
64^3	5,260	2.01	32,342	7.85	6.62	84.36	0.2030	2.57	10.094
128^3	20,943	1.00	70,034	29.20	23.37	80.06	3.0822	7.80	47.218
256^3	83,518	0.50	176,584	172.33	111.12	64.48	49.0320	17.09	224.981

Table 5.2: Statistics for a fluid simulation around the Bunny model (5,002 triangles) on grids of various resolutions. For each grid resolution, the table provides the number of cut cells (# Cut-Cells), its *percentage* with respect to the total number of cubic cells in the regular grid (% Cut-Cells), the number of triangles in the Bunny mesh after intersecting with the grid (# Polygons), the total time in seconds required to generate the cut cells (CC), the subset of the cut-cell generation time spent on CGAL operations (CGAL (sec)), the *percentage* of the cut-cell generation time corresponding to CGAL operations (CGAL (%)), the time spent by our meshing algorithm in the key step of finding all faces incident on each mixed node (MN), and the advection and projection times for simulating one time step.

flow mixing from different sides of a thin boundary. Section 5.2 presents scenarios where objects with complex boundary meshes are embedded in a regular grid. The challenge in these scenarios is to create conforming flows that do not interpenetrate irregularly-shaped objects without heavy reliance on collision detection. Lastly, Section 5.3 discusses flows through narrow regions or gaps, where standard regular-grid solvers would be unable to topologically model the domain.

5.1 Flow around thin objects

Flows with the presence of infinitesimally thin embedded geometries face the challenge of isolating different flow characteristics inside a same regular-grid cell. Our results show that the proposed method in this thesis is able to prevent flow mixing from different sides of a thin object. On the next subsections, we present the following examples: 2-D diagonal line with free-slip boundary conditions, 2-D oscillating lines, 2-D stirring line, 3-D disk slicing through smoke and 2-D/3-D rotating line/paddle.

5.1.1 Diagonal Line

We illustrate the accuracy of the ideal free-slip conditions with a 2-D example of a diagonal solid line embedded in a perfectly tangential flow. The 2-D line remains still for some time, then it starts rotating in alternating directions. Our accompanying video and the image sequence shown in Figure 5.1 show that the line does not disrupt the flow unless it rotates.

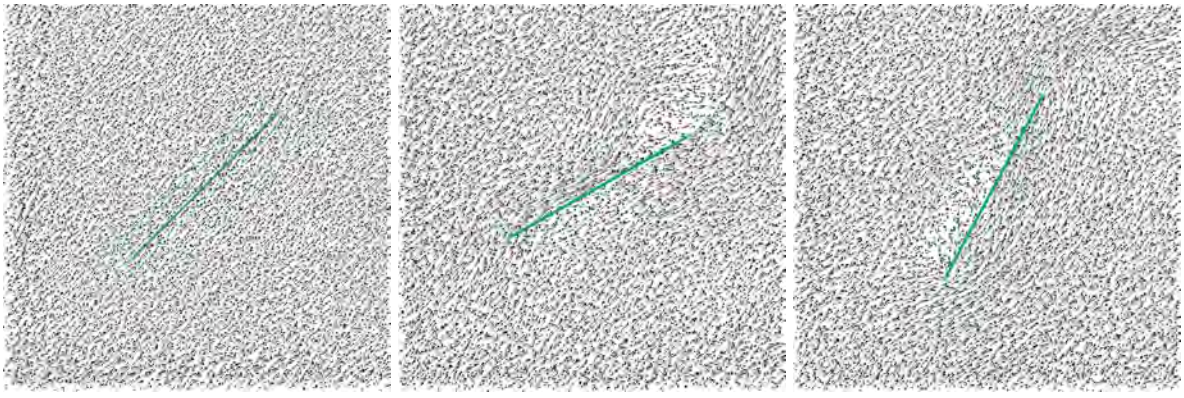


Figure 5.1: Different frames for the diagonal line example. Left image: line remains still, and the diagonal flow is undisturbed by its presence because of free-slip boundary conditions. Center image: line starts to rotate clockwise, disturbing the flow. Right image: line rotates in counter-clockwise direction, also disturbing the flow.

5.1.2 Oscillating Lines

A sequence of examples in our video and in Figure 5.2 feature two vertical line segments oscillating back and forth horizontally. At the closest point of their trajectories, the segments occupy the same column of grid cells, dividing them into three sub-cells (left-most images of Figure 5.2); even in this extreme case the flow behaves naturally. We can see all our contributions in action: the cut-cell pressure projection yields proper fluxes, our PIC/FLIP particles ensure coherent motion without velocity extrapolation for swept-over regions, and our modified interpolation yields particle motion that conforms closely to segments. A close-up illustrates that with free-slip (bottom-right image of Figure 5.2), the fluid flows vertically even as it is squeezed out of the slender sub-cell narrow gap at the closest position. With no-slip (top-right image of Figure 5.2), the interpolated vertical velocities in the gap drop to zero when the segments enter the same grid column, since they are forced to match that the solid.

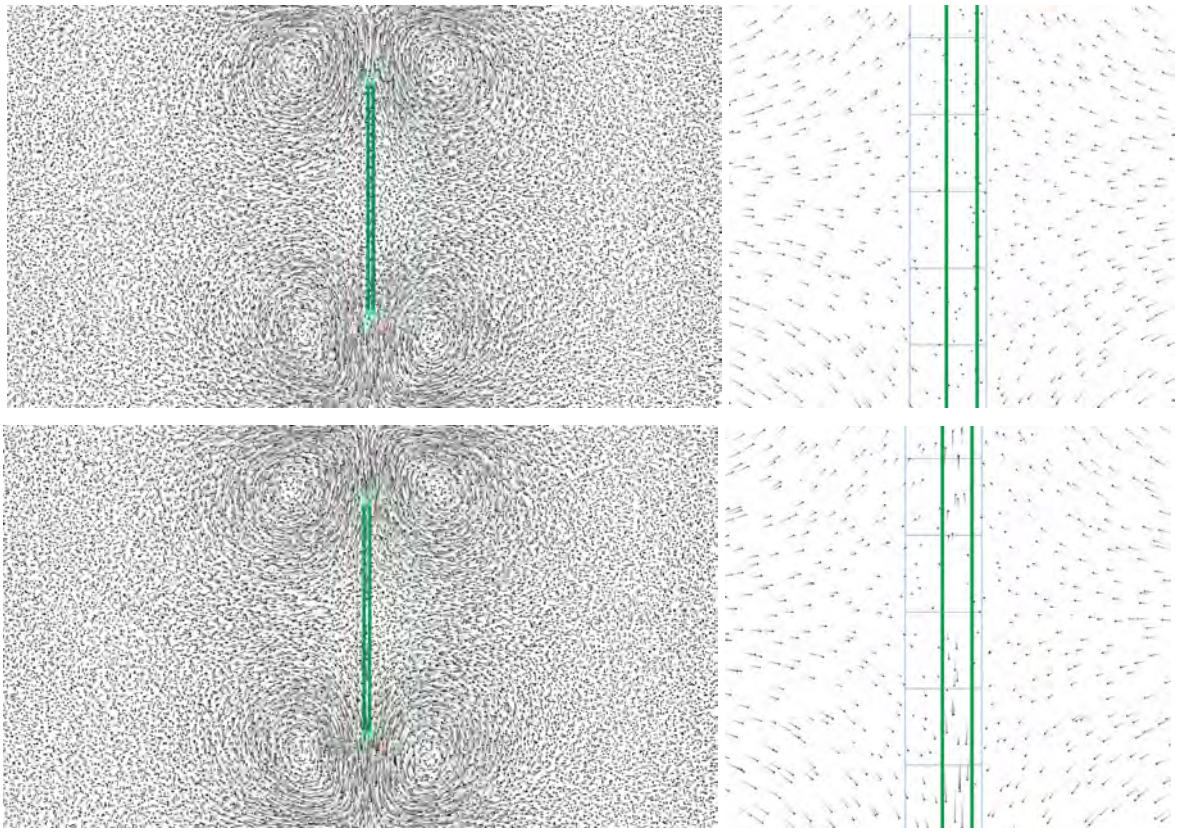
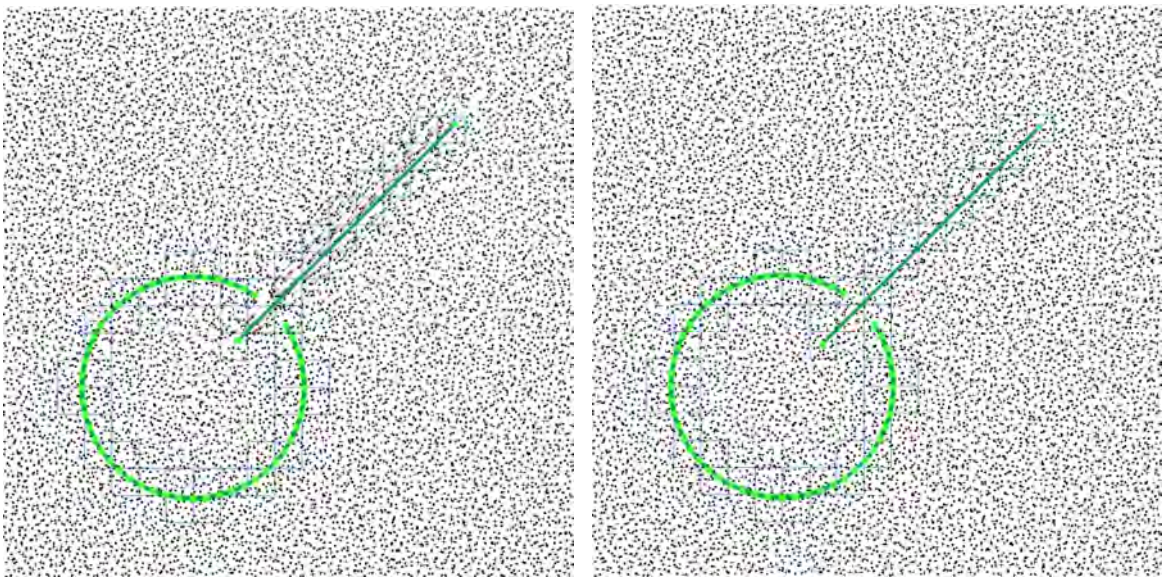


Figure 5.2: Oscillating lines examples: no-slip boundary conditions (top), and free-slip boundary conditions (bottom). Images on the right are the zoomed version for the current time-step of the images on the left. In the free-slip case, particles can freely move up and down through the gap between thin objects.

5.1.3 Stirring Line

In Figure 5.3b, we show a free-slip test in a related scenario where the object is translating tangentially without disrupting the flow; later it begins to rotate and stir the surrounding fluid. In Figure 5.3a the same example is reproduced with no-slip boundary conditions, which immediately induce flow as the line translates tangentially.



(a) (b)

Figure 5.3: No-slip and free slip stress tests. In this example the line translates tangentially to its geometry: In (a), with no-slip the translation disrupts the flow, since the velocity at the boundary is set to be the solid's velocity. In (b), with free-slip boundaries, the fluid is freely allowed to flow tangentially and no disturbance is shown, as expected.

5.1.4 Disk Slicing Smoke

Extending the above scenario to 3-D, we reproduce a test proposed by Robinson-Mosher et al. (ROBINSON-MOSHER; ENGLISH; FEDKIW, 2009) in which a disk with infinitesimal thickness slices tangentially through a block of stationary smoke (Figure 5.4). With ideal free-slip, the smoke should remain perfectly stationary as the disk slips through edge-on; when it passes through a second time while rotating, the smoke should be disturbed. The accuracy of the cut-cell pressure solver allows our simulator to pass this stress test, in contrast to the results in previous work.

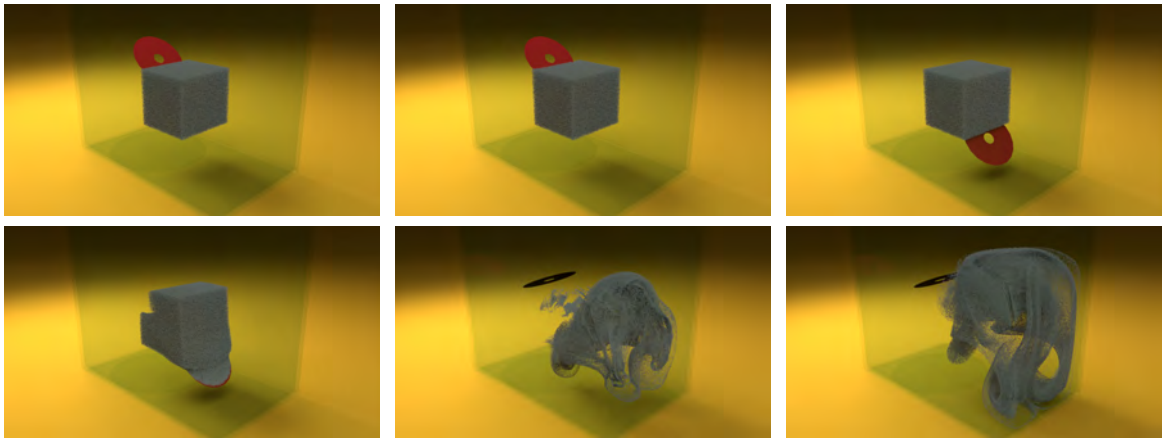


Figure 5.4: A disk passing through smoke, first tangentially (2nd column), then while rotating (3rd column). Top row: Free-slip case. The smoke is undisturbed after the first tangential slice through. Bottom row: No-slip case. The smoke is disturbed immediately.

5.1.5 Rotating Line and Paddle

Figure 5.5 shows a 2-D example of a rotating line with no-slip conditions, demonstrating that the fluid is able to faithfully react to and follow the moving geometry. In the 3-D variant of this example shown in Figure 5.6, a rotating thin paddle translates back and forth through a fluid domain stirring up a volume of smoke on a grid with dimensions $23 \times 10 \times 6$. This example is modeled after one proposed by Klingner et al. (KLINGNER et al., 2006) and used by Batty et al. (BATTY; BERTAILS; BRIDSON, 2007), with the exception that our paddle is extremely thin relative to the grid resolution. This compares favourably to the work of Batty et al. who used as their lowest resolution a grid of $40 \times 20 \times 30$ in order to ensure that their much thicker paddle was adequately resolved at the grid scale. In this example, to avoid issues caused by inadequate treatment of dangling interior faces provided by Shepard interpolation, we assigned the paddle a finite but very small thickness which ensures that SBC is used.

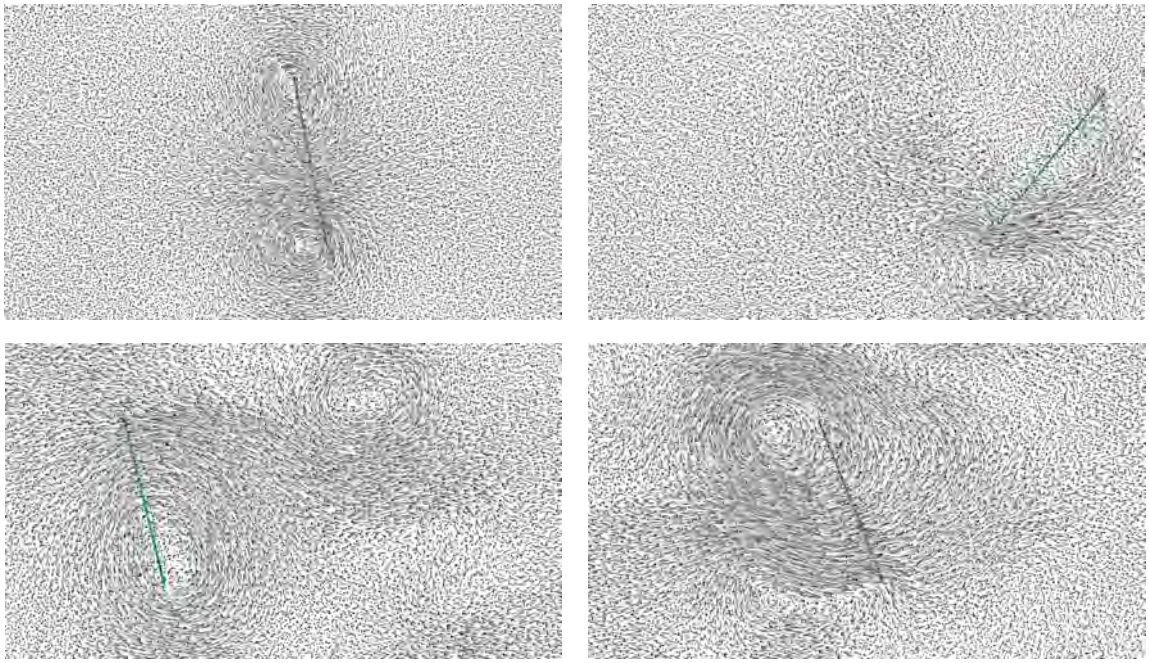


Figure 5.5: Frames from rotating line example for 2-D.

5.2 Flow around irregular objects

Irregularly-shaped objects embedded in regular-grid settings seldom conform with regular grid cells in flow animation scenarios, and have to be handled explicitly. Our results enhance the state-of-the-art boundary treatment for these scenarios, as we are able to reproduce approximately boundary conforming streamlines for both no-slip and free-slip boundary conditions on irregular objects. The following scenarios are discussed in this section: 2-D smoothed gear, 2-D three circles, 3-D dragon and 3-D bunny.

5.2.1 Smoothed Gear

Figure 5.7 shows a irregularly shaped object immersed in a regular-grid setting. The boundary conditions are set to free-slip, allowing tangential flow to conform the irregular boundaries of the object. A rotational velocity field is enforced on the outer part of the gear at the beginning of the simulation. In inner part, no flow is present, since our algorithm is able to successfully isolate the inner and outer regions. Notice that the cut-cells (outlined in blue) are coarse relative to the geometry, creating cut-cells that are curvilinear. Even though, our interpolation scheme is able to maintain accurate tangential free slip flow streamlines that do not penetrate the object.

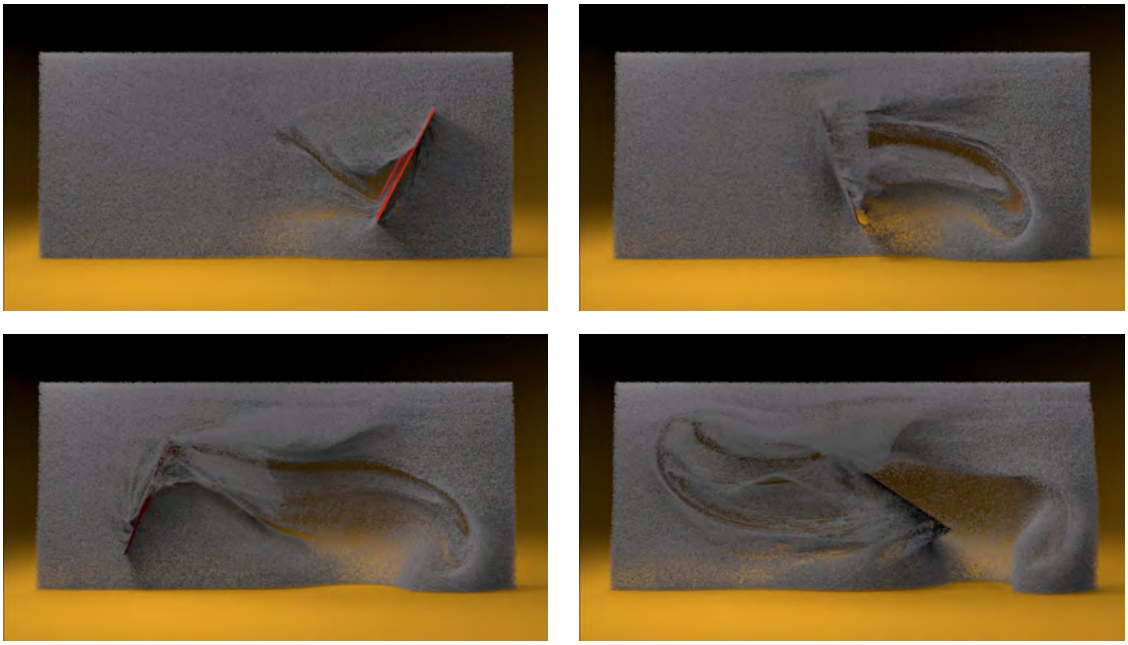


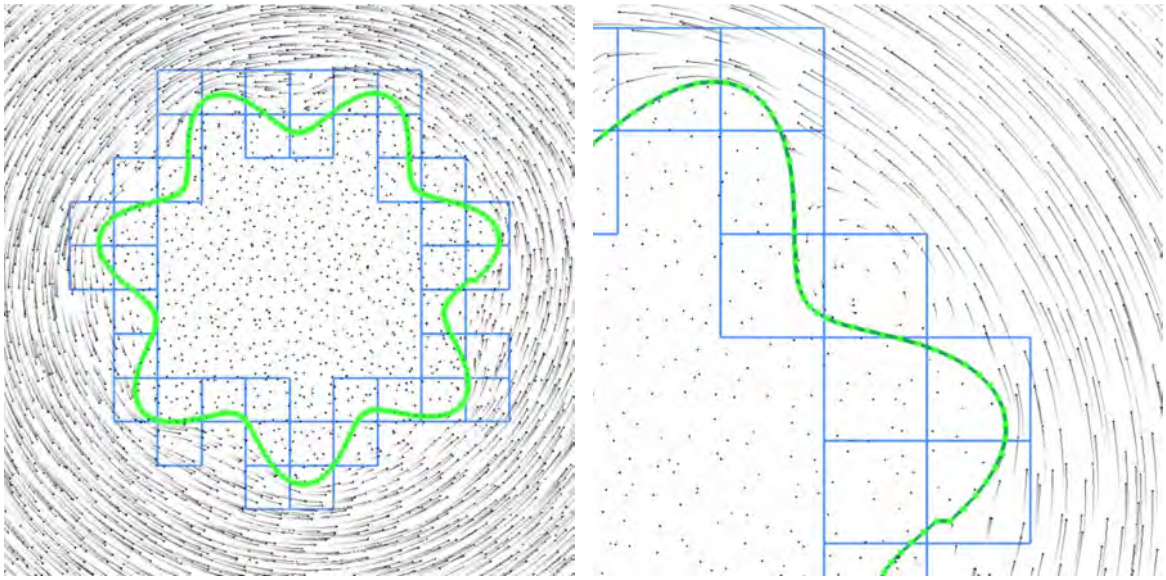
Figure 5.6: Frames from rotating paddle example for 3-D.

5.2.2 Three Circles

Figure 5.8a shows an example of topologically complex scenario coupled with objects with curved boundaries. Three circles are placed close to each other, creating narrow communicating gaps. If a standard regular grid approximation were used, no flow would pass between the circles, since pressure information connecting the flux coming from the left into the bottom and right parts of the flow would not be available. Figure Figure 5.8b shows the same example, this time zoomed in the gaps between the objects.

5.2.3 Bunny

Figure 5.9 illustrates the improved quality of flow around coarse 3-D objects (5,002 triangles) as well. Conforming polyhedral interpolation ensures a reasonable motion that follows the curves of the bunny even on a $7 \times 7 \times 7$ grid. Trading computational cost for quality, Figure 5.10 shows how the level of turbulent detail increases with higher resolution grids.



(a) (b)

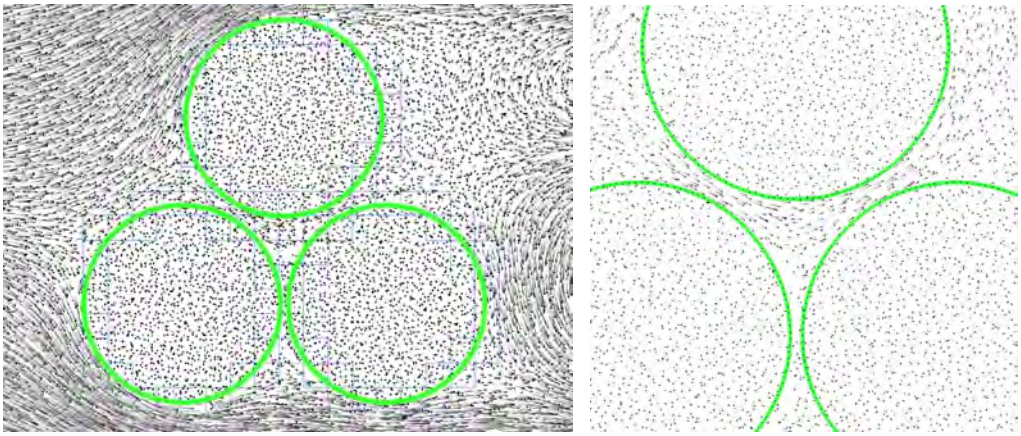
Figure 5.7: Irregularly shaped object example: (a) Gear-like shape with high frequency features, (b) zoomed version of the same example.

5.3 Flow through narrow regions

We consider narrow regions those which have distinct mesh patches inside a single regular-grid cell. In scenarios like this, pressure samples are connected through a topological graph that enables flows to reach in small spaces of the simulation domain. The following flow scenarios are discussed in this section: 2-D branching tubes, 2-D maze, 3-D dragon and 3-D linked tori.

5.3.1 Linked Tori

Figure 5.11 shows a 3-D two linked tori (grid dimensions: $7 \times 7 \times 7$) with flows through quite narrow regions discretized by only a few cells. Velocity forcing is provided by setting a velocity condition on a few faces inside each torus. Moreover, our free-slip implementation allow particles to flow closely to the torus geometry. However, we observed that a very small percentage of the animated particles crosses the object boundary. This happens because our free-slip method projects penetrating velocity components using normals defined at vertices, thus when particles are really close to faces, particles are able to cross the object boundary. We avoid crossing by using a collision detection algorithm for particles that are really close to the object's boundary mesh.



(a) (b)

Figure 5.8: Three circles example. (a) Three circles with a small gap between them; (b) zoomed version of the three circles experiment.

5.3.2 Branching Tube

Figure 5.12 shows a challenging example of flow in a thin gap: a narrow tube, whose width is less than that of a grid cell, turns and branches, creating numerous cut cells and a rich topology. This illustrates the ability of our technique to handle flows through complex regions defined by closely spaced thin boundaries. Results are shown for interpolation using both free-slip (top) and no-slip (bottom) rules. Here and elsewhere free-slip is generally preferred for its superior behavior, but we show various examples with no-slip for completeness.

5.3.3 Flows in a Maze

In this example we show a flow inside a maze. The flow is able to topologically solve the maze (finding the entrance and the exit of it), as expected in a real case scenario. Notice that in Figure 5.13a the maze is highly coarse (blue lines), and all the flow is topologically flowing through cut-cells. We can also see that the velocity interpolation is consistent even through highly non-convex cells. In Figure 5.13b we show a "ghost" geometry moving through a maze with double the resolution of the previous example. The flow also respects the maze topology, while it shows challenging cut-cells shapes being resolved accurately.

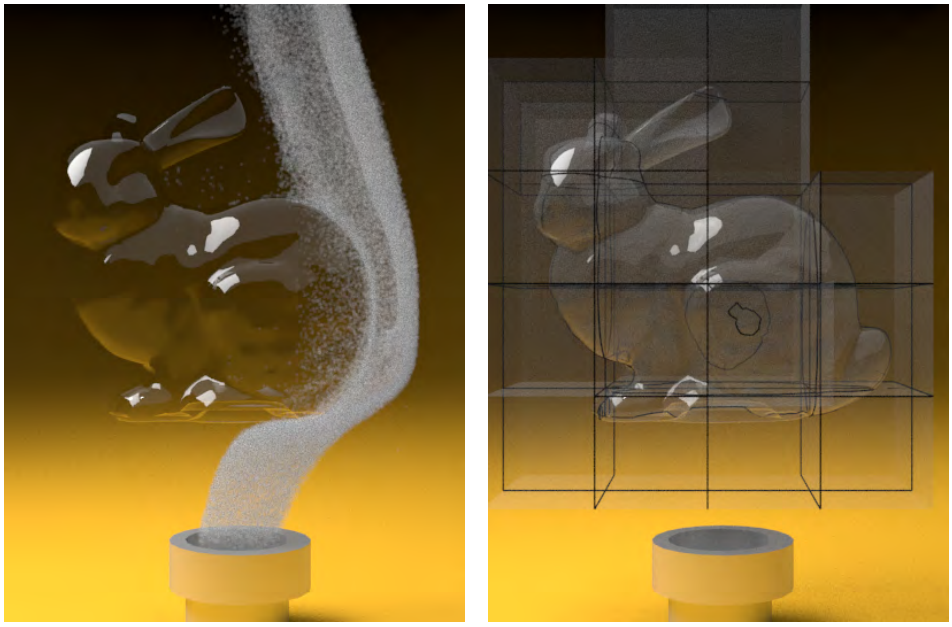
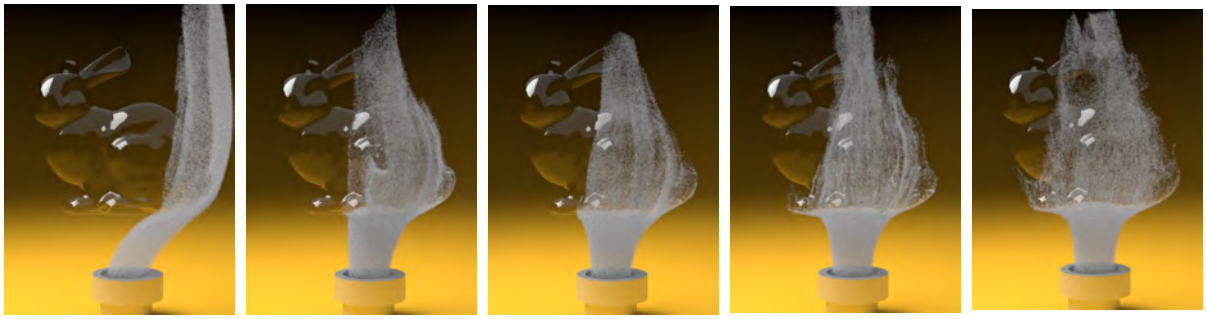


Figure 5.9: Left: The fluid flow conforms to the irregular bunny mesh due to our use of conforming polyhedral interpolation. Right: The same bunny with black curves illustrating the coarse grid.



$8 \times 8 \times 8$ $16 \times 16 \times 16$ $32 \times 32 \times 32$ $64 \times 64 \times 64$ $128 \times 128 \times 128$

Figure 5.10: Simulation of fluid flow around the Bunny model (free-slip, same time step) using grids of various resolutions. While the level of turbulent detail naturally increases at higher resolutions, the flow still respects the geometry even at extremely coarse resolutions.

5.3.4 Dragon

Figure 5.14 shows a complex dragon mesh, where a smoke stream passes through narrow regions inside the dragon mesh. Velocity forcing is provided by setting a velocity condition on a few faces at the tail of the dragon. We ran a second version this same experiment at approximately twice the grid resolution in each dimension (Figure 5.15); while the flow is indeed smoother at finer resolution, it flows in essentially the same fashion as its less well-resolved counterpart. It is this tradeoff of quality and cost, independent of the geometry, that we seek to provide the user.

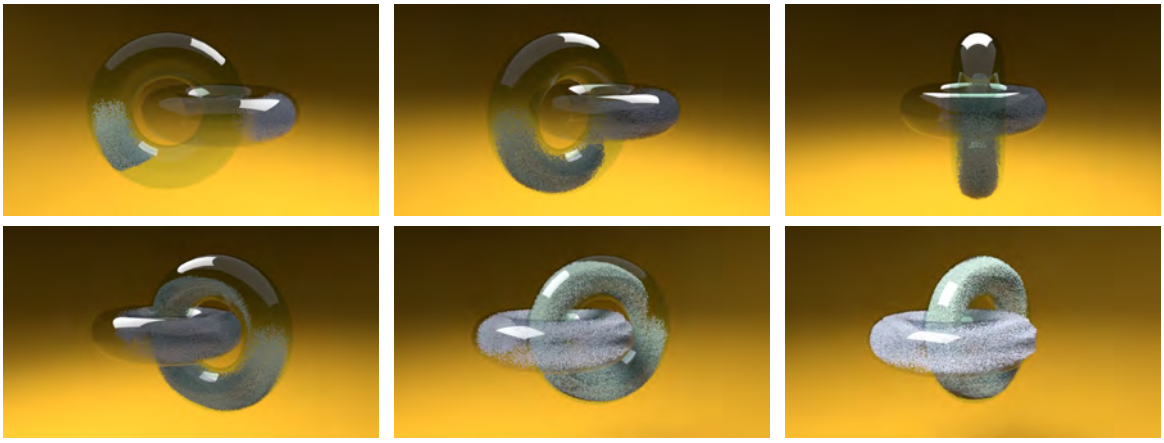
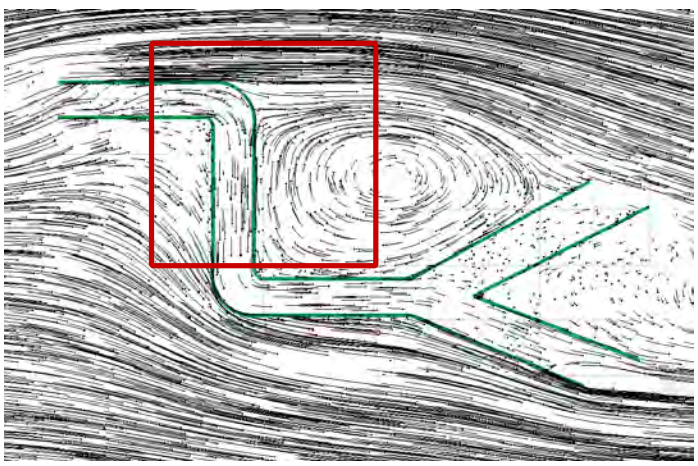
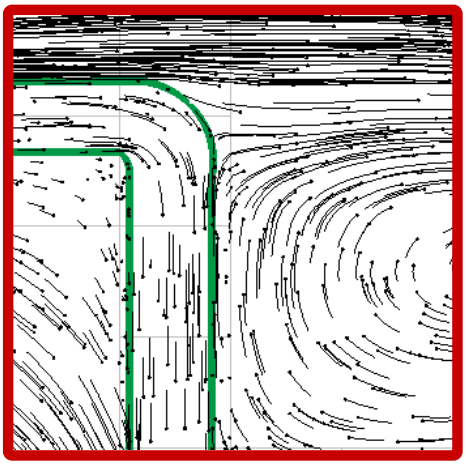


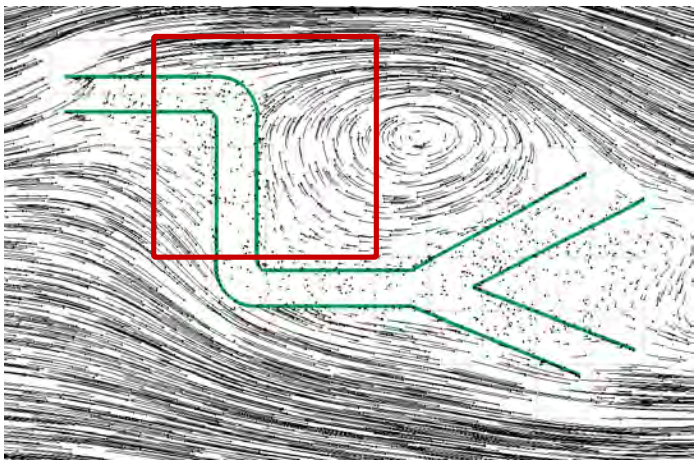
Figure 5.11: Frames from the linked torii example.



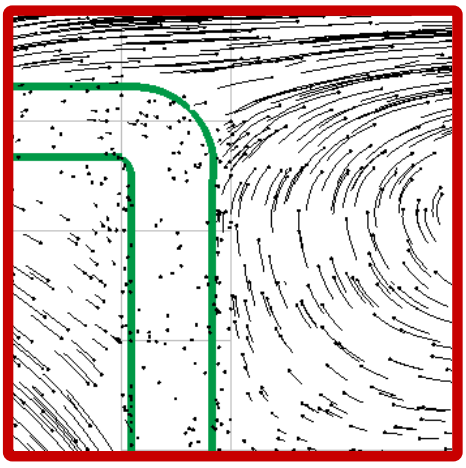
(a) Free-slip



(b) Free-slip Closeup

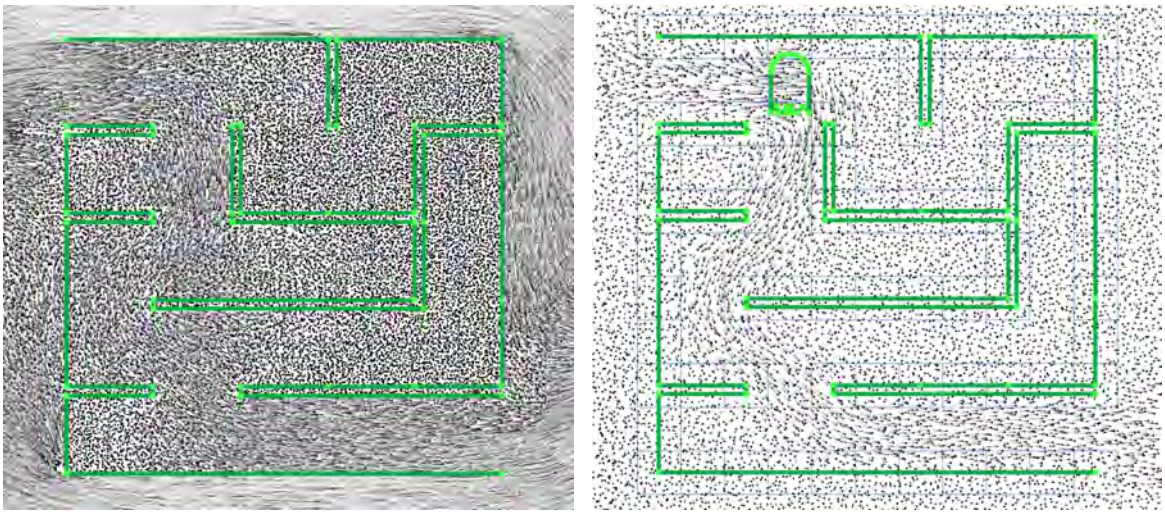


(c) No-slip



(d) No-slip Closeup

Figure 5.12: Flow simulation on a turning and branching tube whose width is smaller than a grid cell width. (a) Free-slip flows smoothly. (c) No-slip halts in the tube. (b) and (d) show closeup views of the highlighted regions.



(a) (b)

Figure 5.13: Example of flow inside a simple maze. The flow is able to topologically solve the maze, as expected in a real case scenario. a) The flow is presented with an increased number of FLIP particles to visualize how the interpolation process is consistent across sharp corners and highly non-convex cells. b) "Ghost" moving through the maze also disrupts the flow in a way that converges to the maze solution.

Figure 5.14: Frames of 3-D dragon animation discretized with a regular grid spacing of $h = 0.5$.

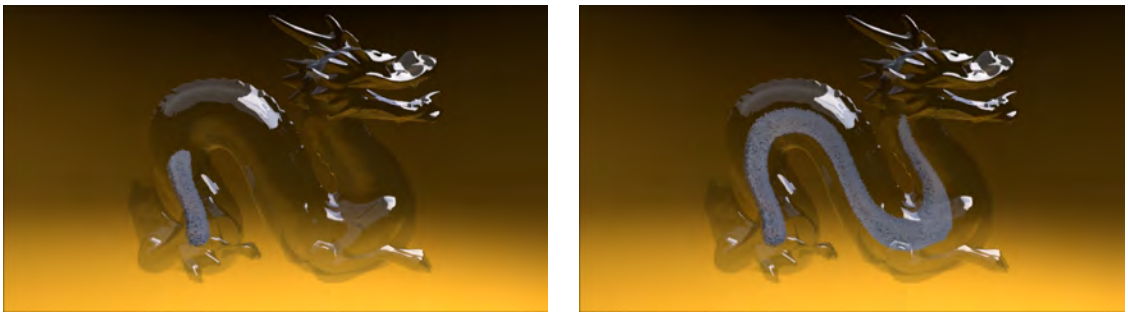


Figure 5.15: Frames of 3-D dragon animation discretized with a regular grid spacing of $h = 0.25$.

6 CONCLUSION

This thesis introduced a technique for simulating fluids in the presence of infinitesimally thin obstacles, narrow gaps, and complex shapes. Our cut-cell meshes are constructed only on the immediate vicinity of any embedded mesh, and as our results show, add little overhead to the overall simulation time. A symmetric positive definite pressure matrix formulation was employed to model the problem and second-order accuracy was obtained for the solution of the pressure system. Our advection scheme is able to produce boundary conforming streamlines and improves current velocity treatment for moving boundaries. Such improvements enabled us to simulate flows in challenging scenarios - such as objects in close proximity and highly irregular shapes - on very coarse grids settings. The concepts presented in this thesis can be readily applied for industry applications, such as creating a computationally fast and topologically correct preview of flows interacting with complex scenes while using very coarse grids.

6.1 Limitations

Our pressure discretization ignores dangling interior solid faces arising from *partially* cut cells, as in previous regular grid schemes for thin boundaries (e.g., (DAY et al., 1998; GUENDELMAN et al., 2005; ROBINSON-MOSHER; ENGLISH; FEDKIW, 2009)). Precisely accounting for this geometry would require generating a fully unstructured *conforming* mesh within the cell. While coupling a regular grid MAC scheme to a full FEM scheme is possible (e.g., (ZHENG et al., 2015)), it would sacrifice the numerical and implementation benefits of our chosen *nearly-regular* grid discretization.

When tunnels between solid geometry within a single cell become *very* small or labyrinthine, a sufficient number of particles may fail to flow into or through gaps. Hence, truly extreme scenarios, such as flow through stacked pages of a book or pores of a sponge, remain impractical; porous flow or homogenization schemes may be preferable.

Although our interpolation method improves the interpolated velocities and resulting trajectories, it cannot *guarantee* trajectories never cross, due to truncation error in time integration. For free-slip conditions on cells with sharp corners, the interpolated velocity may also still have trajectories that do not satisfy $\mathbf{u}_{fluid} \cdot \mathbf{n} = \mathbf{u}_{solid} \cdot \mathbf{n}$ at all points along the perimeter of the cell. More broadly, our interpolants cannot ensure *pointwise* divergence-free velocity fields, which can lead to uneven particle distributions or poor flows near high-frequency geometry. However, for open regular grid regions, even standard trilinear and tricubic interpolation do not yield

Lines	# CC	% CC	CC Gen.	Adv.	Proj.	Total
1	8	5.00	0.001	0.0016	0.023	0.0256
8	29	18.12	0.003	0.0028	0.035	0.0408
64	197	123.12	0.038	0.0120	0.063	0.1130

Table 6.1: For one step at a fixed 2D grid resolution (16×10), 1, 8, and 64 lines per cell crossing over 5 cells (as in Figure 3.12 top). # of Cut-Cells (CC), percentage of CC relative to the regular 160 grid cells. Times (sec) for: CC generation, advection, projection, and total time.

pointwise divergence-free fields. This highlights an interesting challenge for future work: can one construct interpolants which simultaneously (a) respect the discrete face velocities from the pressure solve, (b) accurately conform to boundaries, and (c) satisfy pointwise incompressibility? This holds out the potential to produce significantly improved visual results even in extremely under-resolved regions.

6.2 Future Work

Our approach should be readily extensible to two-way coupling. The challenge for this would be to couple the pressure gradient across the solid-fluid interface without changing the symmetric positive definiteness property of our matrix formulation. A similar and related extension would be incorporating our cut-cell method to capture sub-grid free surface flows details. Since our method enables the use of very coarse grid settings, it is a natural candidate for incorporating sub-grid turbulence to incorporate even greater apparent detail on coarse settings.

REFERENCES

AFTOSMIS, M. J.; BERGER, M. J.; MELTON, J. E. Robust and efficient Cartesian mesh generation for component-based geometry. **AIAA Journal**, v. 36, n. 6, p. 952–960, 1998.

ANDO, R.; THUEREY, N.; WOJTAN, C. Highly adaptive liquid simulations on tetrahedral meshes. **ACM Trans. Graph. (SIGGRAPH)**, v. 32, n. 4, p. 103, 2013.

ANDO, R.; THUEREY, N.; WOJTAN, C. A stream function solver for liquid simulations. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 34, n. 4, p. 53:1–53:9, jul. 2015. ISSN 0730-0301. Available from Internet: <http://doi.acm.org/10.1145/2766935>.

AZEVEDO, V. C.; OLIVEIRA, M. M. Efficient smoke simulation on curvilinear grids. **Computer Graphics Forum**, v. 32, n. 7, p. 235–244, October 2013.

BARGTEIL, A. W. et al. A semi-lagrangian contouring method for fluid simulation. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 25, n. 1, p. 19–38, jan. 2006. ISSN 0730-0301. Available from Internet: <http://doi.acm.org/10.1145/1122501.1122503>.

BATTY, C.; BERTAILS, F.; BRIDSON, R. A fast variational framework for accurate solid-fluid coupling. **ACM Trans. Graph. (SIGGRAPH)**, v. 26, n. 3, p. 100, 2007. ISSN 0730-0301. Available from Internet: <http://portal.acm.org/citation.cfm?id=1276502&CFID=4700498&CFTOKEN=64533352>.

BATTY, C.; XENOS, S.; HOUSTON, B. Tetrahedral embedded boundary methods for accurate and flexible adaptive fluids. **Computer Graphics Forum (Eurographics)**, v. 29, n. 2, p. 695–704, 2010. Available from Internet: <http://onlinelibrary.wiley.com/doi/10.1111/j.1467-8659.2009.01639.x/full>.

BERGER, M. J.; LEVEQUE, R. J. An adaptive cartesian mesh algorithm for the euler equations in arbitrary geometries. In: AIAA COMPUTATIONAL FLUID DYNAMICS CONFERENCE. **Proceedings of the AIAA 9th Computational Fluid Dynamics Conference**. Buffalo, NY, USA, 1989. p. 1–7.

BOJSEN-HANSEN, M.; WOJTAN, C. Liquid surface tracking with error compensation. **ACM Trans. Graph. (SIGGRAPH)**, v. 32, n. 4, p. 79:1–79:10, 2013.

BRACKBILL, J.; RUPPEL, H. Flip: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. **Journal of Computational Physics**, Elsevier, v. 65, n. 2, p. 314–343, 1986.

BROCHU, T.; BATTY, C.; BRIDSON, R. Matching fluid simulation elements to surface geometry and topology. **ACM Trans. Graph. (SIGGRAPH)**, v. 29, n. 4, p. 47, 2010. Available from Internet: <http://portal.acm.org/citation.cfm?id=1778784&CFID=4700498&CFTOKEN=64533352>.

BUSARYEV, O. et al. Animating bubble interactions in a liquid foam. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 31, n. 4, p. 63:1–63:8, jul. 2012. ISSN 0730-0301. Available from Internet: <http://doi.acm.org/10.1145/2185520.2185559>.

CGAL. **CGAL User and Reference Manual**. 4.8. ed. CGAL Editorial Board, 2016. Available from Internet: <http://doc.cgal.org/4.8/Manual/packages.html>.

CHENTANEZ, N. et al. Liquid simulation on lattice-based tetrahedral meshes. In: ACM SIGGRAPH/EUROGRAPHICS SYMPOSIUM ON COMPUTER ANIMATION. **Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation**. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2007. p. 219–228. ISBN 978-1-59593-624-4.

CHENTANEZ, N. et al. Simultaneous coupling of fluids and deformable bodies. In: ACM SIGGRAPH/EUROGRAPHICS SYMPOSIUM ON COMPUTER ANIMATION. **Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation**. Vienna, Austria, 2006. p. 83–89.

CHENTANEZ, N.; MULLER, M. Real-time eulerian water simulation using a restricted tall cell grid. In: ACM SIGGRAPH. **Proceedings of 2011 ACM SIGGRAPH**. Vancouver, British Columbia, Canada: ACM, 2011. (SIGGRAPH '11), p. 82:1–82:10. ISBN 978-1-4503-0943-1. Available from Internet: <http://doi.acm.org/10.1145/1964921.1964977>.

CHERN, A. et al. SchrÖdinger's smoke. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 35, n. 4, p. 77:1–77:13, jul. 2016. ISSN 0730-0301. Available from Internet: <http://doi.acm.org/10.1145/2897824.2925868>.

CHOI, J.-I. et al. An immersed boundary method for complex incompressible flows. **Journal of Computational Physics**, Elsevier, v. 224, n. 2, p. 757–784, 2007.

CHORIN, A. J. Numerical solutions of the navier-stokes equations. **Mathematics of computation**, American Mathematical Society, v. 22, n. 104, p. 745–762, October 1968.

CLARKE, D. K.; HASSAN, H.; SALAS, M. Euler calculations for multielement airfoils using cartesian grids. **AIAA journal**, v. 24, n. 3, p. 353–358, 1986.

CLAUSEN, P. et al. Simulating liquids and solid-liquid interactions with Lagrangian meshes. **ACM Trans. Graph.**, v. 32, n. 2, p. 17, 2013.

COHEN, J. M.; TARIQ, S.; GREEN, S. Interactive fluid-particle simulation using translating eulerian grids. In: ACM SIGGRAPH/EUROGRAPHICS SYMPOSIUM ON INTERACTIVE 3D GRAPHICS AND GAMES. **Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games**. New York, NY, USA: ACM, 2010. p. 15–22. ISBN 978-1-60558-939-8.

CROCKETT, R. K.; COLELLA, P.; GRAVES, D. T. A Cartesian grid embedded boundary method for solving the Poisson and heat equations with discontinuous coefficients in three dimensions. **J. Comp. Phys.**, v. 230, n. 7, p. 2451–2469, 2011.

DAY, M. et al. **Embedded boundary algorithms for solving the Poisson equation on Complex Domains**. [S.l.], 1998. Available from Internet: <http://www.seesar.lbl.gov/ccse/Publications/day/dclrm98/dclrm.ps.gz>.

DEZEEUW, D.; POWELL, K. G. An adaptively refined cartesian mesh solver for the euler equations. **Journal of Computational Physics**, Elsevier, v. 104, n. 1, p. 56–68, 1993.

DICK, C.; ROGOWSKY, M.; WESTERMANN, R. Solving the Fluid Pressure Poisson Equation Using Multigrid - Evaluation and Improvements. **IEEE TVCG**, v. 22, n. 11, p. 2480–2492, 2016.

EDWARDS, E.; BRIDSON, R. Detailed water with coarse grids: Combining surface meshes and adaptive discontinuous Galerkin. **ACM Trans. Graph. (SIGGRAPH)**, v. 33, n. 4, p. 136:1–136:9, 2014.

ELCOTT, S. et al. Stable, circulation-preserving, simplicial fluids. **ACM Trans. Graph.**, v. 26, n. 1, p. 4–22, 2007.

ENGLISH, R. E. et al. Chimera grids for water simulation. In: ACM SIGGRAPH/EUROGRAPHICS SYMPOSIUM ON COMPUTER ANIMATION. **Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation**. New York, NY, USA: ACM, 2013. (SCA '13), p. 85–94. ISBN 978-1-4503-2132-7. Available from Internet: <http://doi.acm.org/10.1145/2485895.2485897>.

ENRIGHT, D.; MARSCHNER, S.; FEDKIW, R. Animation and rendering of complex water surfaces. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 21, n. 3, p. 736–744, 2002. ISSN 0730-0301.

ENRIGHT, D. et al. Using the particle level set method and a second order accurate pressure boundary condition for free surface flows. In: ASME-JSME JOINT FLUIDS ENGINEERING CONFERENCE. **Proceedings of the 4th ASME-JSME Joint Fluids Engineering Conference**. Honolulu, HI: ASME, 2003. p. 337–342.

FADLUN, E. et al. Combined immersed-boundary finite-difference methods for three-dimensional complex flow simulations. **Journal of computational physics**, Elsevier, v. 161, n. 1, p. 35–60, 2000.

FAUCI, L. J.; PESKIN, C. S. A computational model of aquatic animal locomotion. **Journal of Computational Physics**, v. 77, n. 1, p. 85 – 108, 1988. ISSN 0021-9991. Available from Internet: <http://www.sciencedirect.com/science/article/pii/0021999188901581>.

FEDKIW, R.; STAM, J.; JENSEN, H. W. Visual simulation of smoke. In: ACM SIGGRAPH/EUROGRAPHICS SYMPOSIUM ON INTERACTIVE 3D GRAPHICS AND GAMES. **Proceedings of the 28th annual conference on Computer Graphics and Interactive Techniques**. [S.l.], 2001. p. 15–22.

FEDKIW, R. P. et al. A non-oscillatory eulerian approach to interfaces in multimaterial flows (the ghost fluid method). **J. Comput. Phys.**, Academic Press Professional, Inc., San Diego, CA, USA, v. 152, n. 2, p. 457–492, jul. 1999. ISSN 0021-9991. Available from Internet: <http://dx.doi.org/10.1006/jcph.1999.6236>.

FELDMAN, B. E.; O'BRIEN, J. F.; KLINGNER, B. M. Animating gases with hybrid meshes. **ACM Trans. Graph. (SIGGRAPH)**, ACM Press, v. 24, n. 3, p. 904–909, jul. 2005. ISSN 0730-0301. Available from Internet: <http://portal.acm.org/citation.cfm?id=1073281>.

FELDMAN, B. E. et al. Fluids in deforming meshes. In: SIGGRAPH/EUROGRAPHICS SYMPOSIUM OF COMPUTER ANIMATION. **Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium of Computer Animation**. [S.l.], 2005. p. 255–259. ISBN 1-7695-2270-X.

FERSTL, F.; WESTERMANN, R.; DICK, C. Large-scale liquid simulation on adaptive hexahedral grids. **IEEE transactions on visualization and computer graphics**, IEEE, v. 20, n. 10, p. 1405–1417, 2014.

FOGELSON, A. L. A mathematical model and numerical method for studying platelet adhesion and aggregation during blood clotting. **Journal of Computational Physics**, v. 56, n. 1, p. 111 – 134, 1984. ISSN 0021-9991. Available from Internet: <http://www.sciencedirect.com/science/article/pii/002199918490086X>.

FOSTER, N.; FEDKIW, R. Practical animation of liquids. In: ACM SIGGRAPH/EUROGRAPHICS SYMPOSIUM ON INTERACTIVE 3D GRAPHICS AND GAMES. **Proceedings of the 28th annual conference on Computer Graphics and Interactive Techniques**. [S.l.], 2001. p. 23–30. ISBN 1-58113-374-X.

FOSTER, N.; METAXAS, D. Modeling the motion of a hot, turbulent gas. In: SIGGRAPH '97. **Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques**. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997. p. 181–188. ISBN 0-89791-896-7. Available from Internet: <http://dx.doi.org/10.1145/258734.258838>.

GIBOU, F. et al. A second-order-accurate symmetric discretization of the poisson equation on irregular domains. **Journal of Computational Physics**, Elsevier, v. 176, n. 1, p. 205–227, 2002.

GOLDSTEIN, D.; HANDLER, R.; SIROVICH, L. Direct numerical simulation of turbulent flow over a modeled riblet covered surface. **Journal of Fluid Mechanics**, v. 302, p. 333–376, 11 1995. ISSN 1469-7645. Available from Internet: <http://journals.cambridge.org/article_S0022112095004125>.

GORNAK, T. **A goal oriented survey on immersed boundary methods**. [S.l.], 2013.

GUENDELMAN, E. et al. Coupling water and smoke to thin deformable and rigid shells. **ACM Transactions on Graphics**, New York, NY, USA, v. 24, n. 3, p. 973–981, 2005. ISSN 0730-0301.

GUÉZIEC, A. et al. Cutting and stitching: Converting sets of polygons to manifold surfaces. **IEEE Transactions on Visualization and Computer Graphics**, IEEE, v. 7, n. 2, p. 136–151, 2001.

HELLRUNG, J. et al. A second order virtual node method for elliptic problems with interfaces and irregular domains. **J. Comp. Phys.**, v. 231, n. 4, p. 2015–2048, 2012.

HONG, J.-M.; SHINAR, T.; FEDKIW, R. Wrinkled flames and cellular patterns. In: ACM SIGGRAPH. **Proceedings of ACM SIGGRAPH 2007**. San Diego, California: ACM, 2007. (SIGGRAPH '07). Available from Internet: <http://doi.acm.org/10.1145/1275808.1276436>.

HOU, G.; WANG, J.; LAYTON, A. Numerical methods for fluid-structure interaction — a review. **Communications in Computational Physics**, v. 12, p. 337–377, 8 2012. ISSN 1991-7120. Available from Internet: <http://journals.cambridge.org/article_S1815240600003029>.

HOUSTON, B.; BOND, C.; WIEBE, M. A unified approach for modeling complex occlusions in fluid simulations. In: ACM SIGGRAPH. **Proceedings of SIGGRAPH Sketches**. San Diego, California, USA, 2003.

IHMSEN, M. et al. Sph fluids in computer graphics. In: ACM SIGGRAPH/EUROGRAPHICS SYMPOSIUM ON INTERACTIVE 3D GRAPHICS AND GAMES. **Eurographics 2014 - State of the Art Reports**. Strasbourg, France: The Eurographics Association, 2014. p. 21–42. ISSN 1017-4656.

JOHANSEN, H.; COLELLA, P. A Cartesian grid embedded boundary method for Poisson's equation on irregular domains. **J. Comp. Phys.**, v. 147, n. 1, p. 60–85, nov. 1998.

JU, T.; SCHAEFER, S.; WARREN, J. Mean value coordinates for closed triangular meshes. **ACM Trans. Graph. (SIGGRAPH)**, v. 24, n. 3, p. 561–566, 2005.

KIM, D.; SONG, O.-y.; KO, H.-S. Stretching and wiggling liquids. **ACM Trans. Graph.**, v. 28, n. 5, p. 120, 2009.

KIM, J.; KIM, D.; CHOI, H. An immersed-boundary finite-volume method for simulations of flow in complex geometries. **Journal of Computational Physics**, Elsevier, v. 171, n. 1, p. 132–150, 2001.

KIM, T. et al. Wavelet turbulence for fluid simulation. In: ACM SIGGRAPH. **Proceedings of ACM SIGGRAPH 2008 Papers**. Los Angeles, California: ACM, 2008. (SIGGRAPH '08), p. 50:1–50:6. ISBN 978-1-4503-0112-1. Available from Internet: <http://doi.acm.org/10.1145/1399504.1360649>.

KLINGNER, B. M. et al. Fluid animation with dynamic meshes. **ACM Trans. Graph. (SIGGRAPH)**, v. 25, n. 3, p. 820–825, 2006.

LABELLE, F.; SHEWCHUK, J. R. Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles. **ACM Trans. Graph.**, v. 26, n. 3, p. 57, 2007.

LANGER, T.; BELYAEV, A.; SEIDEL, H.-P. Spherical barycentric coordinates. In: EUROGRAPHICS SYMPOSIUM ON GEOMETRY PROCESSING. **Proceedings of the Fourth Eurographics Symposium on Geometry Processing**. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2006. (SGP '06), p. 81–88. ISBN 3-905673-36-3. Available from Internet: <http://dl.acm.org/citation.cfm?id=1281957.1281968>.

LEE, L.; LEVEQUE, R. J. An immersed interface method for incompressible navier-stokes equations. **SIAM J. Sci. Comput.**, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, v. 25, n. 3, p. 832–856, mar. 2003. ISSN 1064-8275. Available from Internet: <http://dx.doi.org/10.1137/S1064827502414060>.

LENAERTS, T.; DUTRÉ, P. **Unified SPH model for fluid-shell simulations**. Celestijnenlaan 200A, 3001 Heverlee, Belgium, 2008. 7 p. Available from Internet: <http://www.cs.kuleuven.be/publicaties/rapporten/cw/CW530.abs.html>.

LENTINE, M.; ZHENG, W.; FEDKIW, R. A novel algorithm for incompressible flow using only a coarse grid projection. In: ACM SIGGRAPH. **Proceedings of 2010 ACM SIGGRAPH**. Los Angeles, California: ACM, 2010. (SIGGRAPH '10), p. 114:1–114:9. ISBN 978-1-4503-0210-4. Available from Internet: <http://doi.acm.org/10.1145/1833349.1778851>.

LI, Z.; LAI, M.-C. The immersed interface method for the navier–stokes equations with singular forces. **Journal of Computational Physics**, v. 171, n. 2, p. 822 – 842, 2001. ISSN 0021-9991. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S0021999101968135>.

LIU, X.-D.; FEDKIW, R. P.; KANG, M. A boundary condition capturing method for poisson's equation on irregular domains. **Journal of Computational Physics**, Elsevier, v. 160, n. 1, p. 151–178, 2000.

LOSASSO, F.; FEDKIW, R.; OSHER, S. Spatially adaptive techniques for level set methods and incompressible flow. **Computers & Fluids**, v. 35, n. 10, p. 995–1010, 2005.

LOSASSO, F.; GIBOU, F.; FEDKIW, R. Simulating water and smoke with an octree data structure. **ACM Trans. Graph. (SIGGRAPH)**, ACM Press, v. 23, n. 3, p. 457–462, aug. 2004. ISSN 0730-0301. Available from Internet: <http://portal.acm.org/citation.cfm?id=1015745>.

MACKLIN, M.; MULLER, M. Position based fluids. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 32, n. 4, p. 104:1–104:12, jul. 2013. ISSN 0730-0301. Available from Internet: <http://doi.acm.org/10.1145/2461912.2461984>.

MARTIN, D. **Solving Poisson's equation using adaptive mesh refinement**. [S.l.]: Citeseer, 1996.

MCQUEEN, D. M.; PESKIN, C. S.; YELLIN, E. L. Fluid dynamics of the mitral valve: physiological aspects of a mathematical model. **American Journal of Physiology - Heart and Circulatory Physiology**, American Physiological Society, v. 242, n. 6, p. H1095–H1110, 1982. Available from Internet: <http://ajpheart.physiology.org/content/242/6/H1095>.

MISZTAL, M. et al. Optimization-based fluid simulation on unstructured meshes. In: VRIPHYS. **Proceedings of the 7th Workshop on Virtual Reality Interaction and Physical Simulation**. [S.l.], 2010.

MITTAL, R. et al. A versatile sharp interface immersed boundary method for incompressible flows with complex boundaries. **Journal of Computational Physics**, v. 227, n. 10, p. 4825 – 4852, 2008. ISSN 0021-9991. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S0021999108000235>.

MITTAL, R.; IACCARINO, G. Immersed boundary methods. **Annual review of fluid mechanics**, Annual Reviews, v. 37, p. 239–261, 2005. ISSN 0066-4189.

MOHD-YUSOF, J. Combined immersed-boundary/b-spline methods for simulations of flow in complex geometries. **Annual Research Briefs. NASA Ames Research Center Stanford University Center of Turbulence Research: Stanford**, p. 317–327, 1997.

MOLEMAKER, J. et al. Low viscosity flow simulations for animation. In: ACM SIGGRAPH/EUROGRAPHICS SYMPOSIUM ON COMPUTER ANIMATION. **Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation**. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2008. p. 9–18. ISBN 978-3-905674-10-1.

MOLINO, N.; BAO, Z.; FEDKIW, R. A virtual node algorithm for changing mesh topology during simulation. **ACM Trans. Graph. (SIGGRAPH)**, ACM Press, v. 23, n. 3, p. 385–392, aug. 2004. ISSN 0730-0301. Available from Internet: <http://portal.acm.org/citation.cfm?id=1015706.1015734>.

MULLER, M.; CHARYPAR, D.; GROSS, M. Particle-based fluid simulation for interactive applications. In: **Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation**. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2003. (SCA '03), p. 154–159. ISBN 1-58113-659-5. Available from Internet: <http://dl.acm.org/citation.cfm?id=846276.846298>.

NESME, M. et al. Preserving topology and elasticity for embedded deformable models. **ACM Trans. Graph. (SIGGRAPH)**, v. 28, n. 3, p. 52, 2009.

NG, Y. T.; MIN, C.; GIBOU, F. An efficient fluid-solid coupling algorithm for single-phase flows. **J. Comp. Phys.**, v. 228, n. 23, p. 8807–8829, 2009. ISSN 0021-9991.

OZGEN, O. et al. Underwater cloth simulation with fractional derivatives. **ACM Transactions on Graphics (TOG)**, ACM, New York, NY, USA, v. 29, n. 3, p. 1–9, 2010. ISSN 0730-0301.

PESKIN, C. S. Flow patterns around heart valves: A numerical method. **Journal of Computational Physics**, v. 10, n. 2, p. 252 – 271, 1972. ISSN 0021-9991. Available from Internet: <http://www.sciencedirect.com/science/article/pii/0021999172900654>.

PESKIN, C. S. Numerical analysis of blood flow in the heart. **Journal of Computational Physics**, v. 25, n. 3, p. 220 – 252, 1977. ISSN 0021-9991. Available from Internet: <http://www.sciencedirect.com/science/article/pii/0021999177901000>.

PURVIS, J. W.; BURKHALTER, J. E. Prediction of critical Mach number for store configurations. **AIAA Journal**, v. 17, n. 11, p. 1170–1177, 1979.

QIU, L.; FEDKIW, R. On thin gaps between rigid bodies two-way coupled to incompressible flow. **J. Comput. Phys.**, Academic Press Professional, Inc., San Diego, CA, USA, v. 292, n. C, p. 1–29, jul. 2015. ISSN 0021-9991. Available from Internet: <http://dx.doi.org/10.1016/j.jcp.2015.03.027>.

QIU, L.; LU, W.; FEDKIW, R. An adaptive discretization of compressible flow using a multitude of moving cartesian grids. **Journal of Computational Physics**, v. 305, p. 75 – 110, 2016. ISSN 0021-9991. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S0021999115006890>.

QUIRK, J. J. An alternative to unstructured grids for computing gas dynamic flows around arbitrarily complex two-dimensional bodies. **Computers & Fluids**, v. 23, n. 1, p. 125 – 142, 1994. ISSN 0045-7930. Available from Internet: <http://www.sciencedirect.com/science/article/pii/0045793094900310>.

RASMUSSEN, N. et al. Directable photorealistic liquids. In: ACM SIGGRAPH/EUROGRAPHICS SYMPOSIUM ON COMPUTER ANIMATION. **Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation**. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2004. (SCA '04), p. 193–202. ISBN 3-905673-14-2. Available from Internet: <http://dx.doi.org/10.1145/1028523.1028549>.

ROBINSON-MOSHER, A.; ENGLISH, R. E.; FEDKIW, R. Accurate tangential velocities for solid fluid coupling. In: ACM SIGGRAPH. **Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation**. New Orleans, Louisiana: ACM, 2009. (SCA '09), p. 227–236. ISBN 978-1-60558-610-6. Available from Internet: <http://doi.acm.org/10.1145/1599470.1599500>.

ROBINSON-MOSHER, A.; SCHROEDER, C.; FEDKIW, R. A symmetric positive definite formulation for monolithic fluid structure interaction. **J. Comp. Phys.**, v. 230, n. 4, p. 1547–1566, 2011. Available from Internet: <http://physbam.stanford.edu/~avir/papers.html>.

ROBINSON-MOSHER, A. et al. Two-way coupling of fluids to rigid and deformable solids and shells. **ACM Trans. Graph.**, v. 27, n. 3, p. 46, 2008. Available from Internet: <http://portal.acm.org/citation.cfm?id=1399504.1360645>.

ROBLE, D.; ZAFAR, N. b.; FALT, H. Cartesian grid fluid simulation with irregular boundary voxels. In: ACM SIGGRAPH. **Proceedings of ACM SIGGRAPH 2005 Sketches**. Los Angeles, California: ACM, 2005. Available from Internet: <http://doi.acm.org/10.1145/1187112.1187279>.

ROSATTI, G.; CESARI, D.; BONAVENTURA, L. Semi-implicit, semi-Lagrangian modelling for environmental problems on staggered Cartesian grids with cut cells. **J. Comp. Phys.**, v. 204, n. 1, p. 353–377, mar. 2005. ISSN 00219991. Available from Internet: <http://dx.doi.org/10.1016/j.jcp.2004.10.013>.

SAIKI, E.; BIRINGEN, S. Numerical simulation of a cylinder in uniform flow: Application of a virtual boundary method. **Journal of Computational Physics**, v. 123, n. 2, p. 450 – 465, 1996. ISSN 0021-9991. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S0021999196900364>.

SCHWARTZ, P. et al. A Cartesian grid embedded boundary method for the heat equation and Poisson's equation in three dimensions. **J. Comp. Phys.**, San Diego, CA, USA, v. 211, n. 2, p. 531–550, 2006.

SELLE, A. et al. An unconditionally stable MacCormack method. **SIAM J. Sci.Comput.**, v. 35, n. 2-3, p. 350–371, 2008. ISSN 0885-7474. Available from Internet: <http://portal.acm.org/citation.cfm?id=1401764>.

SHELLEY, M. J.; ZHANG, J. Flapping and bending bodies interacting with fluid flows. **Annual Review of Fluid Mechanics**, v. 43, n. 1, p. 449–465, 2011. Available from Internet: <http://dx.doi.org/10.1146/annurev-fluid-121108-145456>.

SHEPARD, D. A two-dimensional interpolation function for irregularly-spaced data. In: **ACM '68 Proceedings of the 1968 23rd ACM national conference**. [S.l.: s.n.], 1968. p. 517–524.

SIFAKIS, E.; DER, K. G.; FEDKIW, R. Arbitrary cutting of deformable tetrahedralized objects. In: ACM SIGGRAPH/EUROGRAPHICS SYMPOSIUM ON COMPUTER ANIMATION. **Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation**. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2007. p. 73–80. ISBN 978-1-59593-624-0. Available from Internet: <http://dl.acm.org/citation.cfm?id=1272690.1272701>.

SOLENTHALER, B.; PAJAROLA, R. Predictive-corrective incompressible sph. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 28, n. 3, p. 40:1–40:6, jul. 2009. ISSN 0730-0301. Available from Internet: <http://doi.acm.org/10.1145/1531326.1531346>.

STAM, J. Stable fluids. In: ACM SIGGRAPH. **Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques**. Los Angeles, CA: ACM Press/Addison-Wesley Publishing Co., 1999. p. 121–128. ISBN 0-201-48560-5. Available from Internet: <http://dx.doi.org/10.1145/311535.311548>.

STOMAKHIN, A. et al. A material point method for snow simulation. **ACM Transactions on Graphics (TOG)**, ACM, v. 32, n. 4, p. 102, 2013.

STOMAKHIN, A. et al. Augmented mpm for phase-change and varied materials. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 33, n. 4, p. 138:1–138:11, jul. 2014. ISSN 0730-0301. Available from Internet: <http://doi.acm.org/10.1145/2601097.2601176>.

TERAN, J. et al. Creating and simulating skeletal muscle from the visible human data set. **IEEE TVCG**, v. 11, n. 3, p. 317–328, 2005.

THUREY, N. et al. A multiscale approach to mesh-based surface tension flows. In: ACM SIGGRAPH. **Proceedings of ACM SIGGRAPH 2010 Papers**. Los Angeles, California: ACM, 2010. (SIGGRAPH '10), p. 48:1–48:10. ISBN 978-1-4503-0210-4. Available from Internet: <http://doi.acm.org/10.1145/1833349.1778785>.

UDAYKUMAR, H. et al. Multiphase dynamics in arbitrary geometries on fixed cartesian grids. **Journal of Computational Physics**, Elsevier, v. 137, n. 2, p. 366–405, 1997.

UDAYKUMAR, H. et al. A sharp interface cartesian grid method for simulating flows with complex moving boundaries. **Journal of Computational Physics**, Elsevier, v. 174, n. 1, p. 345–380, 2001.

UDAYKUMAR, H.; SHYY, W.; RAO, M. Elafint: a mixed eulerian–lagrangian method for fluid flows with complex and moving boundaries. **International journal for numerical methods in fluids**, Wiley Online Library, v. 22, n. 8, p. 691–712, 1996.

WANG, K. et al. Computational algorithms for tracking dynamic fluid–structure interfaces in embedded boundary methods. **International Journal for Numerical Methods in Fluids**, v. 70, n. 4, p. 515–535, 2012.

WANG, Y. et al. An adaptive virtual node algorithm with robust mesh cutting. In: ACM SIGGRAPH/EUROGRAPHICS SYMPOSIUM ON COMPUTER ANIMATION. **Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation**. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2014. p. 77–85. Available from Internet: <http://dl.acm.org/citation.cfm?id=2849517.2849531>.

WEBER, D. et al. A cut-cell geometric multigrid Poisson solver for fluid simulation. **Computer Graphics Forum**, v. 34, n. 2, p. 481–491, 2015.

WOJTAN, C. et al. Physically-inspired topology changes for thin fluid features. **ACM Trans. Graph. (SIGGRAPH)**, v. 29, n. 3, p. 50, 2010.

WYMAN, N. **State of the Art in Grid Generation. CFD Review**. 2001. Http://www.cfdreview.com/article.pl?sid=01/04/28/2131215. Last access, Sept. 2012. Available from Internet: <http://www.cfdreview.com/article.pl?sid=01/04/28/2131215>.

ZHENG, W. et al. A new incompressibility discretization for a hybrid particle MAC grid representation with surface tension. **J. Comp. Phys.**, v. 280, n. 1, p. 96–142, 2015.

ZHU, Y.; BRIDSON, R. Animating sand as a fluid. **ACM Trans. Graph. (SIGGRAPH)**, ACM Press, v. 24, n. 3, p. 965–972, jul. 2005. ISSN 0730-0301. Available from Internet: <http://portal.acm.org/citation.cfm?id=1073298>.