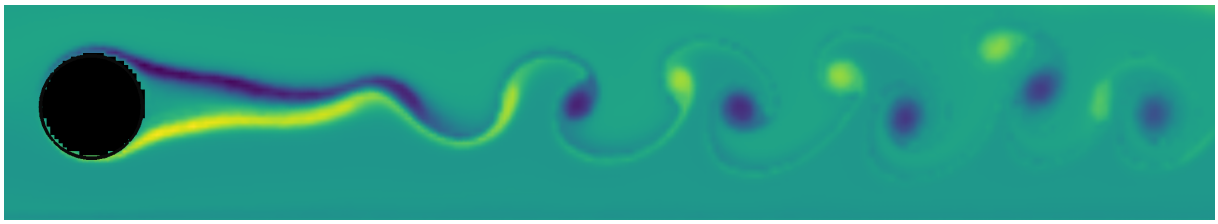
An Efficient Streamfunction-Vorticity Solver for the Simulation of Viscous Fluids



Tom Lausberg

Bachelor Thesis
April 2021

Dr. Vinicius da Costa de Azevedo
Prof. Dr. Markus Gross

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

cgl
computer graphics laboratory

Abstract

In this thesis we investigate a novel approach to solving the Navier-Stokes equations in two dimensions. We focus on the problem of reducing the numerical dissipation which is inherent in many common fluid solvers in computer graphics. To tackle this problem, we solve the streamfunction-vorticity formulation of the Navier-Stokes equation in a divergence free solution space. This allows us to eliminate the dissipative projection step found in pressure-velocity solvers. We discretize the Navier-Stokes equations with discrete exterior calculus, enabling us to build a robust numerical framework to solve our system on non-uniform grids. Finally, we introduce a parametrized viscosity term, which allows us to simulate a wide variety of fluids.

Zusammenfassung

In dieser Arbeit untersuchen wir einen neuartigen Ansatz zur Lösung der Navier-Stokes Gleichungen in zwei Dimensionen. Wir fokussieren uns auf die Reduzierung der numerischen Dissipation, die in viele Fluidsolver inherent ist. Wir bewältigen dieses Problem, indem wir die Streamfunction-Vorticity Formulierung der Navier-Stokes Gleichungen im divergenz-freiem Lösungsraum berechnen. Dies erlaubt es uns, der dissipative Projektionsschritt eines Druck-Geschwindigkeitssolver zu eliminieren. Wir diskretisieren die Navier-Stokes Gleichungen mit Hilfe von "Discrete Exterior Calculus" welches es uns erlaubt, das System auf inhomogene Gitter in einem robusten numerischen Gerüst zu lösen. Letztlich führen wir ein parametrisierter viscocitätsterm ein, die uns ermöglicht, eine breite Vielfalt an Strömungen zu simulieren.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

computer graphics laboratory

Bachelor Thesis

An Efficient Boundary-Respecting Streamfunction-Vorticity Solver

Introduction

Traditional computer graphics methods for simulating fluids often lose accuracy and dissipate in the presence of strong rotational and shearing forces. Previously, we observed that streamfunction-vorticity formulations were able to accurately preserve kinetic energy in fluid solvers. However, dealing with embedded boundary conditions in multiply connected domains require us the calculation of streamfunction values for each embedded object. In this thesis we will extend a streamfunction-vorticity solver to handle arbitrary objects, modelling appropriate boundary conditions with cut-cells. In this way, proper energy conservation is coupled with accurate treatment of boundary conditions, resulting in lively and energetic flow simulations.

Task Description

An existing fluids streamfunction-vorticity fluids framework will be extended to handle cut-cells. The student will first implement a prototype for 2-D; the solution will be subsequently extended for 3-D. A study will be performed on how to efficiently couple cut-cells into the null-space of streamfunction vorticity formulations.

Skills

- Good C++ Skills
- Fluid simulation experience is desired

Remarks

A written report and an oral presentation conclude the thesis. The thesis is will be overseen by Prof. Markus Gross and supervised by Vinicius Azevedo.

Thesis Information

Thesis will be done by Tom Lausberg, from 13/01/2019 to 13/03/2019.

Acknowledgment

I would like to thank my supervisor Dr. Vinicius da Costa de Azevedo for his countless hours of invaluable advice, patience and deep expertise.

In addition, I would like to thank my colleague and friend Safira Piasko, who wrote her bachelor thesis in parallel with mine. Our collaboration in the creation of the base voxelized fluid solver was extremely beneficial to me. I learnt many lessons in our ultimately doomed side quest into the formulation Robin boundary conditions but thoroughly enjoyed the experience.

Contents

List of Figures **xi**

1 Introduction **1**

2 Related Work **3**

3 Mathematical Foundations **5**
3.1 The Navier-Stokes Equations and Streamfunctions 5
3.1.1 Classical Equation 5
3.1.2 Helmholtz-Hodge Decomposition 6
3.1.3 Vorticity-Streamfunction Formulation of the Navier-Stokes Equations . 6
3.1.4 Streamfunction-Vorticity Equations in 2 Dimensions 7
3.2 Boundary Conditions 7
3.2.1 Exterior Boundaries 7
3.2.2 Interior Boundaries 9
3.3 Discrete Exterior Calculus 10
3.3.1 Exterior Derivative 11
3.3.2 Hodge Operator 11
3.3.3 Laplacian Operator 13
3.4 Biharmonic Operator 14
3.5 Diffusion 14
3.5.1 Heat Equation 14
3.5.2 Gaussian Filter 14
3.6 Implementing Viscocity as a Biharmonic Operator 15

4 Implemetation **17**
4.1 Pipeline 17
4.2 Domain 18

4.3 Operator Creation 20
4.3.1 Exterior Derivative 20
4.3.2 Hodge Star 20
4.3.3 Laplace Operator 21
4.4 Boundary Conditions 22
4.4.1 Algorithm 24

5 Results **29**
5.1 Configurations 29
5.2 Examples 30
5.2.1 Empty Channel 30
5.2.2 Object in Channel 30
5.2.3 Object in Channel with diffusion 32
5.2.4 Taylor Vortex 34

6 Conclusion **37**
6.1 Conclusion 37
6.2 Future Research 38

Bibliography **39**

List of Figures

1.1 Water Simulation in animated picture Frozen 2 ©2019 Walt Disney Pictures [PDV] . . . 1

3.1 Cells are mapped to vertices, edges are mapped to edges and vertices are mapped to cells. . . . 12
3.2 Black: Primal Mesh, Blue: Dual Mesh . . . 12
3.3 Discrete Laplace Operator . . . 13

4.1 Overview of the algorithm . . . 17
4.2 Simulation-grid with blue boundary nodes and red interior nodes . . . 19
4.3 Staggered grid with vorticity on nodes and velocity on edges . . . 19
4.4 Voxelized Object . . . 20
4.5 DEC Mesh . . . 21
4.6 Corresponding Delta 1 Operator . . . 21
4.7 Example Mesh . . . 22
4.8 Delta 1 operator . . . 26
4.9 Corrected Delta 1 pperator . . . 26
4.10 Left column: Laplacian, right column: biharmonic operator; top row: original, middle row: corrected, bottom row: right-hand side . . . 27

5.1 Kinetic energy, empty channel . . . 30
5.2 FreeSlip, top: pressure-velocity, bottom: streamfunction-vorticity . . . 31
5.3 NoSlip, top: pressure-velocity, bottom: streamfunction-vorticity . . . 31
5.4 Kinetic energy, object in channel . . . 32
5.5 Object in channel with diffusion, 1st row: $\nu = 0$, 2nd row: $\nu = 0.001$: 3rd row: $\nu = 0.01$, 4th row: $\nu = 0.1$ . . . 33
5.6 Kinetic energy, object in channel with diffusion . . . 34
5.7 Initial vorticity for all simulations . . . 34

5.8 Kinetic energy, taylor vortex 35
5.9 Taylor vortex: 1st row: $\nu = 0$, 2nd row: $\nu = 0.001$: 3rd row: $\nu = 0.01$, 4th row: $\nu = 0.1$ 36

Introduction

Physically based simulation is becoming an ever more important field in computer graphics. Modern films and video games demand complex and realistic representations of our environments which can no longer be manually created with hand animations. Whilst most people are not experts in fluid dynamics, everyone has a strong intuition on how fluids should flow and can easily recognise unrealistic animations. This has driven animation and game studios to use increasingly complex computational fluid dynamic simulations to reach the current level of realism in modern creations. A fundamental problem with CFD simulations is their computational cost. When using CFD one must always balance the runtime of the simulation with the time and cost of hardware needed to run it.

Figure 1.1: *Water Simulation in animated picture Frozen 2 ©2019 Walt Disney Pictures [PDV]*

The fundamental equations of fluid dynamics are the Navier-Stokes equations. The most common approach to solving the Navier-Stokes equations is with a pressure velocity solver. Whilst

this approach has widespread use and many efficient implementations, almost all suffer from inherent numerical dissipation in the pressure projection step. In this thesis we describe a novel approach solving the problem of simulating viscous fluids with the Navier-Stokes equations. We solve the streamfunction-vorticity formulation of the Navier-Stokes equations in which we use the Helmholtz decomposition to enforce divergent free flows. This divergent free method gives us perfectly inviscid flows to which we then add a manually defined viscous term. By manually controlling the viscous term we can precisely simulate a range of different liquids and avoid the difficult process of controlling the numerical dissipation caused by pressure-velocity methods.

Related Work

The accurate and efficient simulation of fluids with numerical methods is a well explored field of computer graphics. This has led to a plethora of different approaches to solving the Navier-Stokes equations [Sto45]. These approaches can be divided into two main groups. Eulerian and lagrangian methods. Eulerian methods view the fluid as a vector field and usually discretize the equations on a regular grid. On the other hand, lagrangian methods describe the fluid flow with particles which carry the computational quantities.

In computer graphics, first eulerian approaches by Foster and Metaxas [FM96] proved efficient grid based methods could be used to render realistic gases and their interaction with solids. This method unfortunately suffered from numerical instabilities. Simultaneously modified smoothed particle hydrodynamics (SPH) methods [DG96] showed the viability of particle based methods. SPH and other lagrangian methods can efficiently calculate the advection of the fluid using simple time stepping on the fluid particle or calculate the fluid interface using the position of said particles. Many of these methods unfortunately suffer from compressibility and lead to diffusive characteristics. On the other hand, eulerian methods offer other advantages and challenges. Whilst they more easily enforce this incompressibility by solving a Poisson equation efficiently on a regular grid, calculating the advection can be much more expensive due to the need to calculate the material derivative. To solve some of these problems, hybrid methods have been proposed. A seminal paper by Harlow [Har] the *Particle in Cell* (PIC) method was first used in computational fluid dynamics. PIC uses lagrangian fluid particles to advect the fluid and transfers their velocities to a grid to then be projected into a divergence free field. This hybrid method was further improved and popularized in computer graphics by the *Fluid Implicit Particle* (FLIP) method [BKR88]. In our method we use a scheme based on both FLIP and PIC as seen in [ZB05].

Whilst valid solutions can efficiently be computed by projecting the flow into a divergence free field, this approach suffers from inherent diffusion. This diffusion is implementation specific and hard to control and correct for. To avoid the problem of pressure projection we use the

streamfunction vorticity formulation of the Navier-Stokes equations [Fro64]. This approach was presented in *"Simulation of Smoke Based on Vortex Filament Primitives"* [AN05] using vortex filaments. Elcoot et. al. [ETK+07] also proposed a grid based solver of the vorticity formulation of the Navier-Stokes equations.

Many concepts in our work are based on the paper *"A Cartesian Grid Method for Solving the two-Dimensional Streamfunction-Vorticity Equations in Irregular Regions"* [Cal02]. This paper also describes a finite difference method for solving the streamfunction-vorticity formulation. The proposed method also handles sub-grid fluid boundaries and explicit handling of the viscosity term.

Recently Valério Sampaio [Sam18] formulated a hybrid FLIP based solver of the streamfunction-vorticity formulation. To simulate fluids in complex domains, the use of the finite difference method and uniform grids can be both expensive and difficult to implement. To handle the computation of partial differential equations on complex meshes, *Discrete Exterior Calculus* (DEC) can be used. Various works in DEC have been summarized by Keenan Crane in the series of notes *"Discrete differential geometry: an applied introduction"* [KC13].

Based on the work by Valério Sampaio and using discrete exterior calculus, Piasko describes a FLIP solver which incorporates subgrid cut-cell boundaries in her thesis *"On Reducing Numerical Dissipation of Euler Equations with Discrete Exterior Calculus"* [Pia20].

Mathematical Foundations

In the following section, we will describe the equations and mathematical constructs needed to create our fluid simulation.

3.1 The Navier-Stokes Equations and Streamfunctions

3.1.1 Classical Equation

The conventional formulation of the Navier-Stokes equation is often written as

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p = \vec{f} + \nabla \cdot \left(\frac{\eta}{\rho} (\nabla \vec{u} + \nabla \vec{u}^T + \lambda (\nabla \cdot \vec{u}) \delta \right) \tag{3.1}$$

[Sto45] Due to the incompressibility of the fluids in our desired simulation, we can remove the final term on the right-hand side of the equation. Because the dynamic viscosity term η is constant, we can further simplify the right-hand side to

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p = \vec{f} + \nu \nabla \cdot \nabla \vec{u} \tag{3.2}$$

$$\nabla \cdot \vec{u} = 0 \tag{3.3}$$

See [Sam18] for more details on the derivation.

3.1.2 Helmholtz-Hodge Decomposition

In our simulation we use the Helmholtz-Hodge Decomposition to restrict the solution space of the velocity field of our fluid. The Helmholtz-Hodge Decomposition a fundamental theorem in the field of fluid simulation. It claims that an arbitrary vector field can be decomposed into divergence free, irrotational and harmonic components.

$$\vec{u} = \underbrace{\nabla\theta}_{\text{gradient of a scalar field}} + \underbrace{\nabla \times \vec{\Psi}}_{\text{curl of a vector field}} + \underbrace{\vec{\gamma}}_{\text{harmonic vector field}} \tag{3.4}$$

Because the fluid is incompressible, the gradient of the scalar field must be zero. Also, we assume the harmonic component of the vector field to be zero. This leads to the simplified expression

$$\vec{u} = \nabla \times \vec{\Psi} \tag{3.5}$$

3.1.3 Vorticity-Streamfunction Formulation of the Navier-Stokes Equations

To derive the Vorticity-Streamfunction Formulation of the Navier-Stokes Equations we first take the curl from each side of the equation and insert the definition of the vorticity $\vec{\omega} = \nabla \times \vec{u}$.

$$\nabla \times \left(\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p \right) = \nabla \times (\vec{f} + \nu \nabla \cdot \nabla \vec{u}) \tag{3.6}$$

$$\frac{\partial}{\partial t} \underbrace{(\nabla \times \vec{u})}_{\text{vorticity } \omega} + \nabla \times ((\vec{u} \cdot \nabla)\vec{u}) + \underbrace{\frac{1}{\rho} \nabla \times \nabla p}_{=0} = \nabla \times \vec{f} + \nu \nabla \times (\nabla \cdot \nabla \vec{u}) \tag{3.7}$$

$$\frac{\partial \vec{\omega}}{\partial t} + \nabla \times ((\vec{u} \cdot \nabla)\vec{u}) = \nabla \times \vec{f} + \nu \nabla \times (\nabla \cdot \nabla \vec{u}) \tag{3.8}$$

$$\frac{\partial \vec{\omega}}{\partial t} + \nabla \times ((\vec{u} \cdot \nabla)\vec{u}) = \nabla \times \vec{f} + \nu \nabla \cdot \nabla \vec{u} \tag{3.9}$$

$$\underbrace{\frac{\partial \vec{\omega}}{\partial t} + (\vec{u} \cdot \nabla)\vec{\omega})}_{\text{material derivative } \frac{D\vec{\omega}}{Dt}} - \underbrace{(\vec{\omega} \cdot \nabla)\vec{u})}_{\text{Vortex stretching}} = \underbrace{\nabla \times \vec{f}}_{\text{external forces}} + \underbrace{\nu \nabla \cdot \nabla \vec{u}}_{\text{viscosity}} \tag{3.10}$$

To relate the vorticity to the streamfunction we can use the definition from equation 3.5.

$$\vec{u} = \nabla \times \vec{\Psi} \tag{3.11}$$

$$\nabla \times \vec{u} = \nabla \times \nabla \times \vec{\Psi} \tag{3.12}$$

$$\vec{\omega} = \nabla(\nabla \cdot \vec{\Psi}) - \nabla \cdot \nabla\vec{\Psi} \tag{3.13}$$

3.1.4 Streamfunction-Vorticity Equations in 2 Dimensions

When applying the equations in two dimensions we can simplify and reduce them further. Firstly, the equation 3.13 can be simplified because the term $\nabla(\nabla \cdot \vec{\Psi})$ is 0. This means the vorticity simply becomes the negative Laplacian of the streamfunction. Because of the cross product, both the x and y components of the streamfunction and vorticity are also set to 0. Henceforth, we will use the notation $\Psi = \vec{\Psi}_z$ and $\omega = \vec{\omega}_z$ This gives us the equation

$$\omega = -\nabla \cdot \nabla\vec{\Psi} = -\Delta\Psi. \tag{3.14}$$

Another simplification in two dimensions, is that vortex stretching does not exist. This allows us to write equation 3.10 as follows

$$\frac{\partial\omega}{\partial t} + \vec{u} \cdot \nabla\omega = \nabla \times \vec{f} + \nu\nabla \cdot \nabla\vec{u}. \tag{3.15}$$

Equations 3.14 and 3.15 become the fundamental equations in our simulation.

3.2 Boundary Conditions

3.2.1 Exterior Boundaries

There are two situations in-which boundaries occur in our simulation. We will first discuss the exterior boundaries. These are found on the border of our simulation domain. On each boundary element we enforce two conditions. Firstly, a Dirichlet condition on the streamfunction Ψ and secondly a condition which can either be formulated as a Dirichlet condition on the vorticity ω or a Neumann condition on the streamfunction. We formulate the second condition as a Dirichlet condition on the vorticity. On the exterior boundary we implemented four boundary types. To formulate our boundary conditions, we define two vectors relative to the boundary. Firstly $\vec{\tau}$ is tangential to the boundary and $\vec{n}$ is normal boundary.

Free-Slip Condition

Free Slip boundaries are impenetrable boundaries, which have no friction. The no-penetration condition implies that the velocity of the fluid normal to the border is 0. Because the velocity

in a direction is equal to the derivative of the streamfunction in perpendicular direction we can define the dirichlet condition as such:

$$\vec{u}_{\vec{n}}(\vec{x}) = \frac{\partial \Psi(\vec{x})}{\partial \vec{\tau}} = 0 \qquad \forall \vec{x} \in \partial\Omega \tag{3.16}$$

$$\Psi(\vec{x}) = \Psi_c \qquad \forall \vec{x} \in \partial\Omega \tag{3.17}$$

This means the streamfunction along all impenetrable boundaries has a constant value Ψ_c on boundary $\partial\Omega$. The vorticity is defined as the curl of the velocity. On a boundary element in two dimensions this is expressed as

$$\omega = \frac{\partial \vec{u}_{\vec{n}}}{\partial \vec{\tau}} - \frac{\partial \vec{u}_{\tau}}{\partial \vec{n}}. \tag{3.18}$$

Because of the no-penetration condition the normal velocity along the boundary is 0 for all values along the boundary. Hence, the value of $u_{\vec{n}}$ is constant and the tangential derivative is 0.

$$\vec{u}_{\vec{n}}(\vec{x}) = 0 = \text{const.} \qquad \forall \vec{x} \in \partial\Omega \tag{3.19}$$

$$\frac{\partial \vec{u}_{\vec{n}}}{\partial \vec{\tau}} = 0 \tag{3.20}$$

The second term in equation 3.18 is also 0 because free-slip boundaries assert that no shear forces are exerted on the liquid on the boundary. Consequentially the difference in fluid velocity tends to 0 at the boundary.

$$\lim_{h \to 0} \vec{u}_{\vec{n}}(\vec{x}) - \vec{u}_{\vec{n}}(\vec{x} + h\vec{n}) = 0 \qquad \forall \vec{x} \in \partial\Omega \tag{3.21}$$

$$\frac{\partial \vec{u}_{\tau}(\vec{x})}{\partial \vec{n}} = 0 \tag{3.22}$$

This means the vorticity along free-slip boundaries is always 0.

$$\omega(\vec{x}) = 0 \qquad \forall \vec{x} \in \partial\Omega \tag{3.23}$$

No-Slip Condition

For the no-slip condition we once again have the no-penetration condition. Therefore, the Dirichlet condition is also

$$\Psi(\vec{x}) = \Psi_c \qquad \forall \vec{x} \in \partial\Omega \tag{3.24}$$

The Dirichlet condition on the vorticity is also determined by equation 3.18. Here the tangential derivative of normal velocity is also zero, which leads to

$$\omega = \underbrace{\frac{\partial \vec{u}_{\vec{n}}}{\partial \vec{\tau}}}_{=0} - \frac{\partial \vec{u}_{\tau}}{\partial \vec{n}}. \tag{3.25}$$

The term that changes the normal derivative of the tangential velocity. Here we must enforce that the tangential velocity is 0 at the boundary. Using equation 3.11 we formulate this condition as

$$\vec{u}_\tau = \frac{\partial \Psi}{\partial \vec{n}} = 0. \tag{3.26}$$

We now focus on computing the second term of equation 3.25

$$\frac{\partial \vec{u}_\tau}{\partial \vec{n}} = -\frac{\partial^2 \Psi}{\partial \vec{n}^2}. \tag{3.27}$$

For this we first take the Taylor-expansion of the streamfunction near the boundary $\partial\Omega$ and isolate the second derivative on the left-hand side.

$$\Psi(\vec{x} + h\vec{n}) = \Psi(\vec{x}) + h \underbrace{\frac{\partial \Psi}{\partial \vec{n}}}_{=0} + \frac{h^2}{2}\frac{\partial^2 \Psi}{\partial \vec{n}^2} + \mathcal{O}(h^3) \qquad x \in \partial\Omega \tag{3.28}$$

$$\frac{\partial^2 \Psi}{\partial \vec{n}^2} \approx -2\frac{\Psi(\vec{x} + h\vec{n}) - \Psi(\vec{x})}{h^2}. \tag{3.29}$$

By inserting this definition and 3.28 in equation 3.25 we get

$$\omega(\vec{x}) = -\frac{\partial \vec{u}_\tau}{\partial \vec{n}} = 2\frac{\Psi(\vec{x} + h\vec{n}) - \Psi(\vec{x})}{h^2} \qquad x \in \partial\Omega. \tag{3.30}$$

Inflow

For inflow boundaries the only constraint is that the flow rate normal to the border is predetermined. This is enforced by fixing the streamfunction such that it's derivative in tangential direction is equal to the inflow speed

$$\vec{u}_{\text{inflow}} = \vec{u}_\tau = \frac{\partial \Psi}{\partial \vec{\tau}}. \tag{3.31}$$

Outflow

On outflow borders no boundary conditions are enforced. These borders are handled similarly to the interior of the simulation domain but use a modified stencil in the Poisson/diffusion step to ensure that they are only dependent on neighboring vertices contained in the simulation domain.

3.2.2 Interior Boundaries

Interior boundaries exist on the border of solid objects inside the computational domain. As with the free and no-slip exterior boundaries, we enforce the no-penetration condition. Consequently,

the Dirichlet condition of the streamfunction on the boundary $\partial\Omega$ is always

$$\Psi(\vec{x}) = \Psi_c \qquad \forall x \in \partial\Omega. \tag{3.32}$$

We also have the same conditions on the vorticity, namely equation 3.23 for free-slip and equation 3.30 for no-slip. The main difference is in how the constant streamfunction value on the boundary must be calculated in each timestep. For exterior boundaries, the streamfunction is a predetermined value which is both constant in time and space. For the interior boundaries, the no-penetration condition enforces that the streamfunction Ψ_c remains constant in space but we must introduce another condition found in [Cal02] to compute its value over time. To do this, we formulate the momentum equation in terms of the tangential velocity

$$\frac{\partial \vec{u}_\tau}{\partial t} + \left(\vec{u}_\tau \frac{\partial}{\partial \vec{\tau}} + \vec{u}_n \frac{\partial}{\partial \vec{n}}\right) \vec{u}_\tau = -\frac{\partial p}{\partial \vec{\tau}} + \nu \frac{\partial \omega}{\partial \vec{n}}. \tag{3.33}$$

Given that the velocity on the boundary is $\vec{u}_\tau = \vec{u}_n = 0$ the equation above reduces to

$$\frac{\partial p}{\partial \vec{\tau}} = \nu \frac{\partial \omega}{\partial \vec{n}}. \tag{3.34}$$

We then integrate along the boundary $\partial\Omega$ from $\vec{x}_0$ to $\vec{x}$

$$p(\vec{x}) = p_0 + \nu \int_{\partial\Omega} \frac{\partial \omega}{\partial \vec{n}} d\tau. \tag{3.35}$$

Because the pressure is single valued on the boundary, i.e $p(\vec{x}) = p_0 \quad \forall \vec{x} \in \partial\Omega$, we obtain

$$\int_{\partial\Omega} \frac{\partial \omega}{\partial \vec{n}} d\tau = 0. \tag{3.36}$$

We will later use this condition to determine the streamfunction on the boundary of solid objects.

3.3 Discrete Exterior Calculus

To discretise the Navier-Stokes equations we use an innovative approach based on the theory of Discrete Exterior Calculus. As the name suggests *Discrete Exterior Calculus* (DEC) is a discrete analogous to the field of Exterior Calculus which itself is a field of differential geometry. For a thorough and well formulated introductions to DEC and its applications see [KC13]. The concepts and derivations in this chapter are based on it.

We use discrete exterior calculus to discretise our simulation because it gives a flexible and robust framework to work on non-uniform meshes and utilise the linear nature of its operators to efficiently and intuitively construct discrete formulations of our equations of in our simulation domain.

In this chapter we will discuss how the discrete versions of fundamental exterior calculus operators are formulated and how they can be used to construct the partial differential equations which describe our simulation.

3.3.1 Exterior Derivative

The exterior derivative is a generalisation of concept of a differential. Whilst the differential of a 0-form is a 1-form which describes all directional derivatives of the 0-form, the exterior derivative can be applied to any k-form and will result in a $(k+1)$-form. To formulate the discrete form of the exterior derivative we use Stokes' theorem.

$$\int_{\Omega} d\alpha = \int_{\partial\Omega} \alpha \tag{3.37}$$

Stokes' theorem states that the integral over the exterior derivative of a k-form in domain Ω can be calculated if the integral of the k-form is known on the boundary of the domain. Discrete k-forms are most easily described as the integral of a continuous k-form over a discrete mesh. Due to this we are able to easily compute the right-hand side of Stokes' theorem.

Discrete Exterior Derivative

If we take $\hat{\alpha}$ as the discrete representation of a k-form α which has been integrated over all k-simplices in our computational domain, then $d\hat{\alpha}$,the exterior derivative thereof, is a $(k+1)$-form which integrates over the derivative of each $(k+1)$-simplex in the domain. For a discrete 0-form $\hat{\alpha}$ (e.g. the streamfunction on discrete nodes in a mesh), the discrete exterior derivative of $\hat{\alpha}$ can be described as

$$(d\alpha)_{e_{ij}} = \int_{n_i}^{n_j} \alpha = \hat{\alpha}_j - \hat{\alpha}_i \tag{3.38}$$

along all edges e_{ij} from node i to node j.

Similarly the discrete exterior derivative of $\hat{\alpha}$ can be calculated for the cells in a mesh. We define a cell c_i with edges $e_i j$ $\{j = 0, ..., k-1\}$. The exterior derivative is then

$$(d\alpha)_{c_i} = \int_{\partial c_i} \alpha = \sum_{j=0}^{k} \int_{e_j} \alpha = \sum_{j=0}^{k} \alpha_j. \tag{3.39}$$

3.3.2 Hodge Operator

The Hodge operator maps a k-form in a n-dimensional domain to its dual $(n-k)$-form. An analogous operation in euclidean space is the mapping from planes to normal vectors. To describe the Hodge operator also known as the Hodge Star ($\star$) in a discrete setting we create a dual to our mesh. An example of this can be seen in figure 3.2. In two dimensions this is done by setting dual vertices on each primal cell center. The edges bordering the primal cells are then mapped to edges connecting to dual vertices. Hence, the cells formed by the dual edges correspond to each primal vertex. These transformations are depicted in figure 3.1. The primal elements are shown in black and their corresponding dual elements in blue.

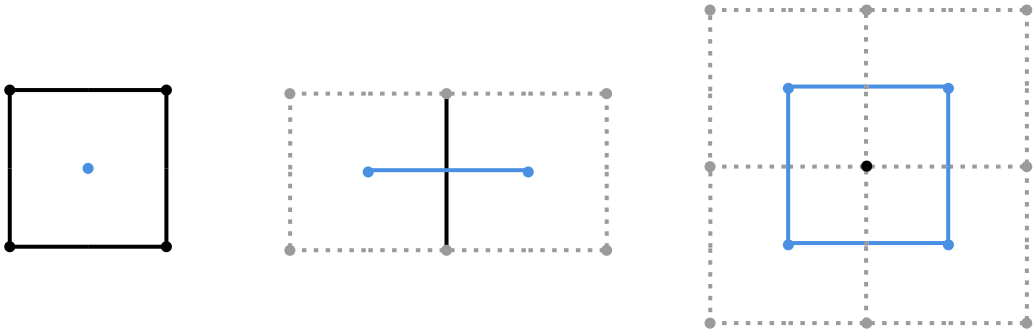


Figure 3.1: *Cells are mapped to vertices, edges are mapped to edges and vertices are mapped to cells.*

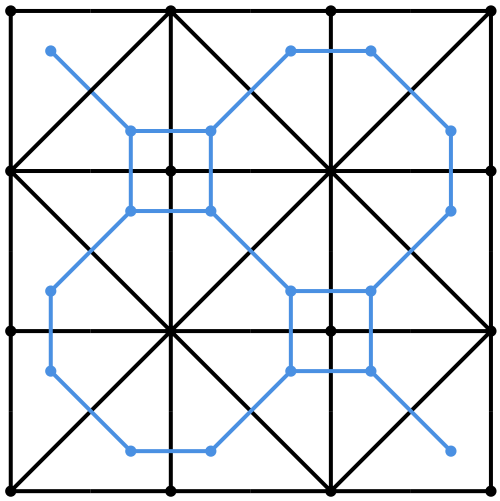


Figure 3.2: *Black: Primal Mesh, Blue: Dual Mesh*

3.3.3 Laplacian Operator

The Laplacian operator is defined as the divergence of the gradient

$$\Delta = \nabla \cdot \nabla. \tag{3.40}$$

The exterior calculus formulation of the Laplacian is as follows:

$$\Delta = \star\, d \star d \tag{3.41}$$

To calculate the Laplacian of a discrete 0-form α on a simulation mesh we can apply each operator sequentially. First, we calculate the exterior derivative of α. With Stokes' theorem we transform it into an integral on the boundary of the edges of our mesh.

$$(d\alpha)_{ij} = \int_{e_{ij}} d\alpha = \int_{\partial e_{ij}} \alpha = \alpha_j - \alpha_i \tag{3.42}$$

We then apply the Hodge operator, transforming the derivative along an edge to its dual. e_{ij} is the edge from vertex i to vertex j and $e^\star_{ij}$ is it's dual.

$$\star\,(d\alpha)_{ij} = \frac{|e^\star_{ij}|}{|e_{ij}|}(\alpha_j - \alpha_i) \tag{3.43}$$

To take the derivative again we use Stokes' theorem and integrate over the dual cell C_i. We can achieve this by summing over all k edges ($e_{ij}\{j = 0, ..., k-1\}$) which are connected to vertex i.

$$d(\star d\alpha)_i = \int_{C_i} d(\star d\alpha) = \int_{\partial C_i} (\star d\alpha) = \int_{\partial C_i} \star(\alpha_j - \alpha_i) = \sum_{j=0}^{k} \frac{|e^\star_{ij}|}{|e_{ij}|}(\alpha_j - \alpha_i) \tag{3.44}$$

Finally, we apply the Hodge operator on $d \star d\alpha$ by dividing by the area of the dual cell C_i.

$$\star\,(d \star d\alpha)_i = \frac{1}{|C_i|} \sum_{j=0}^{k} \frac{|e^\star_{ij}|}{|e_{ij}|}(\alpha_j - \alpha_i) \tag{3.45}$$

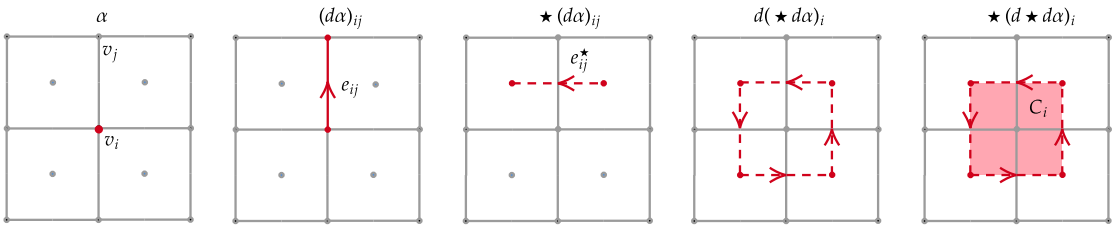


Figure 3.3: *Discrete Laplace Operator*

3.4 Biharmonic Operator

To apply the biharmonic operator to a 0-form α we can simply apply the Laplace operator to the Laplacian of α.

$$\Delta^2\alpha = \Delta(\Delta\alpha) = \star d \star d \star d \star d\alpha. \tag{3.46}$$

3.5 Diffusion

In many circumstances simulating diffusion is vital for creating realistic flows. Pressure-velocity based fluid solver inherently induce numerical diffusion. Whilst this can sometimes be beneficial, numerical diffusion is often unpredictable and hard to control. Because streamfunction-vorticity methods do not suffer from these problems, diffusion must be added explicitly.

3.5.1 Heat Equation

One possibility is to apply the heat equation to either streamfunction or vorticity fields in the system. The diffusion equation is a common parabolic partial differential equation used to model the spread of heat, Brownian motion and many other physical phenomena. The heat equation applied to the vorticity with diffusion coefficient α is expressed as

$$\frac{\partial\omega}{\partial t} = \alpha\Delta\omega \tag{3.47}$$

We can then discretise the temporal derivative using forward differences

$$\frac{\partial\omega}{\partial t} = \alpha\Delta\omega \tag{3.48}$$

$$\frac{\omega^{n+1} - \omega^n}{dt} = \alpha\Delta\omega \tag{3.49}$$

$$\omega^{n+1} - \omega^n = = dt\alpha\Delta\omega \tag{3.50}$$

$$\omega^{n+1} = \omega^n + dt\alpha\Delta\omega \tag{3.51}$$

This modification can be appilied after the advection step. It is also possible to apply a similar diffusion step to the streamfunction after the Poisson step. Unfortunately, both approaches produce unstable solutions and were not further explored.

3.5.2 Gaussian Filter

Another possibility is to apply a gaussian filter over the result of either the streamfunction or the vorticity. Gaussian filters disperse information throughout the system by averaging the value at each node with its neighbors. This can be done by applying the Laplacian to the quantity of interest. For vorticity we compute the following value:

$$\omega' = \Delta\omega \tag{3.52}$$

This too gave us unstable results and thus was not used in our simulations.

3.6 Implementing Viscocity as a Biharmonic Operator

To implement viscosity in our simulation we augment the streamfunction vorticity equation (3.14) with a biharmonic term. In her paper "A Cartesian Grid Method for Solving the Two-Dimensional Streamfunction-Vorticity Equations in Irregular Regions" [Cal02] Calhoun implements viscosity by introducing conditions for stokes flow and formulating a system of equations to solve both the streamfunction and vorticity.

In our system we enforce Stokes' flow using the following constraint

$$\frac{\omega^{n+1} - \omega^*}{dt} = \nu \Delta \omega^{n+1} \tag{3.53}$$

on the vorticity where dt is the timestep, ω^{n+1} denotes the value of the vorticity at time $dt(n+1)$ and ω^* is the vorticity at time $dt \cdot n$ after advection. Our goal is to create a simpler system in which we only solve for the value of the streamfunction. We do this by transforming the equation to isolate ω^{n+1} as such

$$\frac{\omega^{n+1} - \omega^*}{dt} = \nu \Delta \omega^{n+1} \tag{3.54}$$

$$\omega^{n+1} - \omega^* = dt\nu \Delta \omega^{n+1} \tag{3.55}$$

$$\omega^{n+1} - dt\nu \Delta \omega^{n+1} = \omega^* \tag{3.56}$$

$$(\mathbb{1} - dt\nu \Delta)\omega^{n+1} = \omega^*. \tag{3.57}$$

We then replace ω^{n+1} with $-\Delta\Psi$ by using equation 3.14.

$$\omega^{n+1} = -\Delta \Psi^{n+1} \tag{3.58}$$

$$(\mathbb{1} - dt\nu \Delta)(-\Delta \Psi^{n+1}) = \omega^* \tag{3.59}$$

$$(dt\nu \Delta^2 - \Delta)\Psi^{n+1} = \omega^* \tag{3.60}$$

Equation 3.60 now replaces equation 3.14 as a fundamental equation. This formulation is advantageous due to the fact that the system of equations maintains its original dimensions and the new operator $(dt\nu\Delta^2 - \Delta)$ can be easily created by manipulating the original Laplacian operator.

Implemetation

In this chapter we will discuss the specifics of our solver and implementation of the algorithm. Our solver uses a hybrid approach, meaning the advection step is solved in the Lagrangian perspective with particles. The Poisson/diffusion step is then solved on a uniform grid in the Eulerian perspective. We work with three fundamental attributes which are computed in each step. These are the streamfunction Ψ, vorticity ω and velocity $\vec{u}$.

4.1 Pipeline

We break our solver into three main components which are calculated in each time step. These are advection, Poisson/diffusion, and recalculation.

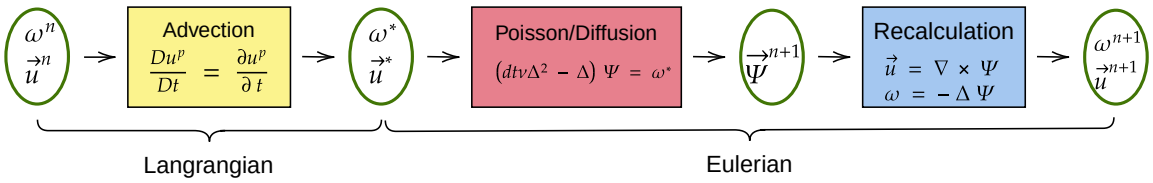


Figure 4.1: *Overview of the algorithm*

Advection

The advection step transports the fluid according to the velocity field. Whilst the Eulerian formulation of the advection equation is

$$\frac{\partial \phi}{\partial t} + \vec{u} \cdot \nabla \phi = 0, \tag{4.1}$$

we make use of a hybrid method simplify its calculation. By solving the advection step in the Lagrangian perspective, we forego this difficult hyberbolic partial differential equation. In the Lagrangian perspective, we can use the FLIP/PIC [ZB05] advection which can be computed with a simple second order Runge Kutta time stepping scheme. Because we switch between the Lagrangian and Eulerian perspective we need interpolations schemes to switch from the grid to the particles and back. For the grid-to-particle interpolation, we use a bilinear interpolation scheme. The step is done at the beginning of each time step and transfers our computed quantities (i.e. $\omega, \vec{u}$) from the nodes and edges of the grid on to implicit particles in our domain. After the particles are advected using the FLIP/PIC scheme, we transfer the values back to the grid using an SPH Kernel [DG96].

Poisson/Diffusion

After the advection step is completed, we have the advected values on the grid which we from here on name auxiliary vorticity and auxiliary velocity. These values are denoted ω^* and $\vec{u}^*$ respectively. The main goal of the Poisson step is to calculate the value of the streamfunction in the next time step. This is done using equation 3.60.

$$(\mathrm{dt}\nu\Delta^2 - \Delta)\Psi^{n+1} = \omega^*$$

The operator $(\mathrm{dt}\nu\Delta^2 - \Delta)$ is only dependant on the mesh, meaning it does not need to be recalculated in every step. To solve the equation efficiently, we represent the operator as a symmetric positive definite matrix in a sparse form. This allows us to solve 3.60 as a linear system of equations using an iterative conjugate gradient solver. We use the Eigen Conjugate Gradient [GJ+10], [HS+52] implementation to do this.

Recalculation

With the values of Ψ^{n+1} we can then calculate the values needed to start the next time step. We calculate the velocity using equation 3.11 and then, after applying boundary conditions, calculate the vorticity by taking the curl of the velocity.

$$\vec{u}^{n+1} = \nabla \times \Psi^{n+1} \tag{4.2}$$

$$\omega^{n+1} = \nabla \times \vec{u}^{n+1} \tag{4.3}$$

4.2 Domain

The computational domain is defined as a rectangle in two dimensions. For the Eulerian part of the solver, the domain is discretized as a uniform grid seen in figure 4.2. We store the value of the streamfunction and the vorticity on the grid vertices. The velocity is represented

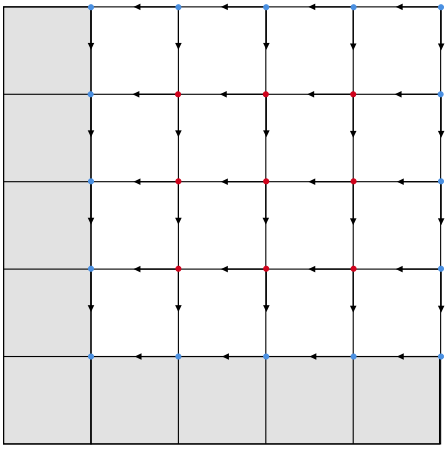


Figure 4.2: *Simulation-grid with blue boundary nodes and red interior nodes*

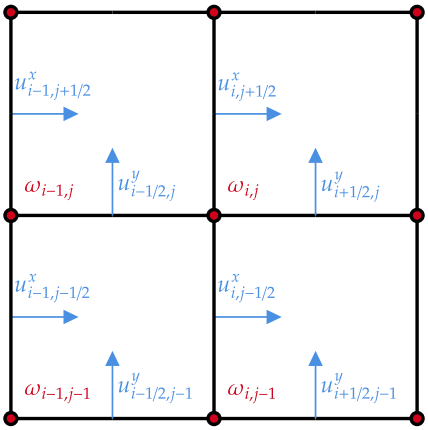


Figure 4.3: *Staggered grid with vorticity on nodes and velocity on edges*

in a staggered form with the x-component on the vertical edges and the y-component of the horizontal edges. A schematic view of quantities' position on the grid can be seen in figure 4.3. The grid is padded along the bottom and left side with ghost cells.

DEC Mesh

The Poisson/diffusion step is done on a subset of the grid called the DEC-Mesh. The DEC-Mesh includes everything but the ghost-cells, shaded gray in figure 4.2. Its task is to create the DEC operators and handle their interaction with the data in the Poisson step.

Solid Objects

We use voxelization to incorporate solid objects in our grid. Voxelized objects are represented by marking all vertices on the edge of a cell containing the object as solid. This is depicted in figure 4.4 with the marked solid vertices shown as black nodes on the grid.

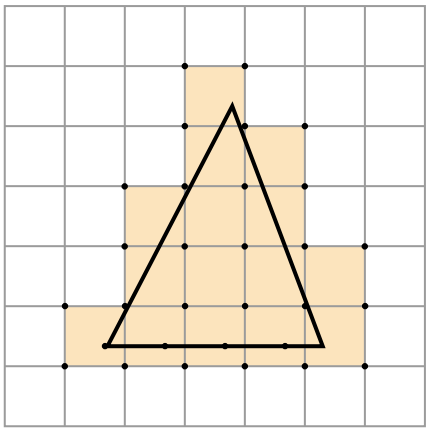


Figure 4.4: *Voxelized Object*

4.3 Operator Creation

4.3.1 Exterior Derivative

To create the Laplacian and biharmonic operators we must first implement the exterior derivative and Hodge star operators. As seen in section 3.3.1, the exterior derivative of a 0-form on the directed edge of a mesh is described as the value of the 0-form on the first vertex subtracted from the second vertex (equation 3.38). We implement this operator as a sparse (#vertices $\times$ #edges)-matrix. Because all edges in a DEC mesh are directional, we can describe edge e_{ij} which connects vertex i to vertex j as the row in the matrix containing the value -1 in the i^{th} column and 1 in the j^{th} column. Doing this for each edge in the matrix gives us the exterior derivative operator (d_1). This operator can then by applied to 0-forms as a matrix-vector multiplication, with the 0-form represented as a vector of length #vertices. An example of a mesh and its corresponding d_1 operator can be seen in figures 4.5 and 4.8.

4.3.2 Hodge Star

To create the Laplacian, we also use the Hodge star operator on two different objects. Firstly, applying the Hodge star of an edge e in two dimensions gives us its dual edge $e^\star$. We implement this operator as a diagonal Matrix of size (#edges $\times$ #edges). The diagonal elements have the values $\frac{|e|}{|e^\star|}$ for all edges in the mesh. Because our mesh is a uniform grid, the lengths of the dual

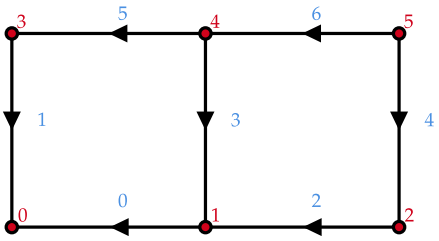


Figure 4.5: *DEC Mesh*

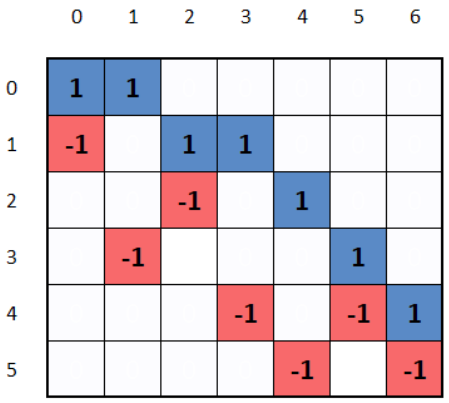


Figure 4.6: *Corresponding Delta 1 Operator*

edges are the same as the lengths of the primal edges.

$$|e| = |e^\star| = \Delta x \qquad \forall e \in \{edges\} \tag{4.4}$$

This means all diagonal elements are 1 and we simply model the Hodge operator on edges as the identity matrix.

$$h_1 = \mathbb{1} \tag{4.5}$$

Similarly, the Hodge operator on the vertices of a mesh can be discretized as a (#vertices $\times$ #vertices)-matrix. The diagonal is given as the inverse of the area of the vertex dual face (see figure 3.1). As this value is always $\frac{1}{(\Delta x)^2}$ for all vertices, the operator is simplified to

$$h_2 = \frac{1}{(\Delta x)^2}\mathbb{1} \tag{4.6}$$

4.3.3 Laplace Operator

Now that we have the necessary DEC operators, we can create the Laplacian.

$$\begin{aligned}
\Delta &= \star\, d \star d && (4.7)\\
&= h_1 d_1 h_2 d_1^T && (4.8)\\
&= \frac{1}{(\Delta x)^2}\mathbb{1} d_1 \mathbb{1} d_1^T && (4.9)\\
&= \frac{1}{(\Delta x)^2} d_1 d_1^T && (4.10)
\end{aligned}$$

This means that the Laplacian operator can simply be created with a single sparse matrix-matrix multiplication. The biharmonic operator is then easily constructed using this Laplacian.

$$\Delta^2 = (\star d \star d)(\star d \star d) \tag{4.11}$$

$$= \frac{1}{(\Delta x)^4}(d_1 d_1^T)^2 \tag{4.12}$$

4.4 Boundary Conditions

As seen in the previous section, it is possible to use DEC to create the operators needed to solve our equations. But before using these operators, we must first incorporate the boundary conditions derived in section 3.2. In the following section, we will explain the procedure used to enforce these boundary conditions. To illustrate the effect of the boundary conditions on the operators, we will use example operators created for the mesh in figure 4.7. The mesh represents a domain with solid boundaries on the top and bottom, an inflow boundary on the left and an outflow boundary on the right. The cell marked in yellow represents a solid.

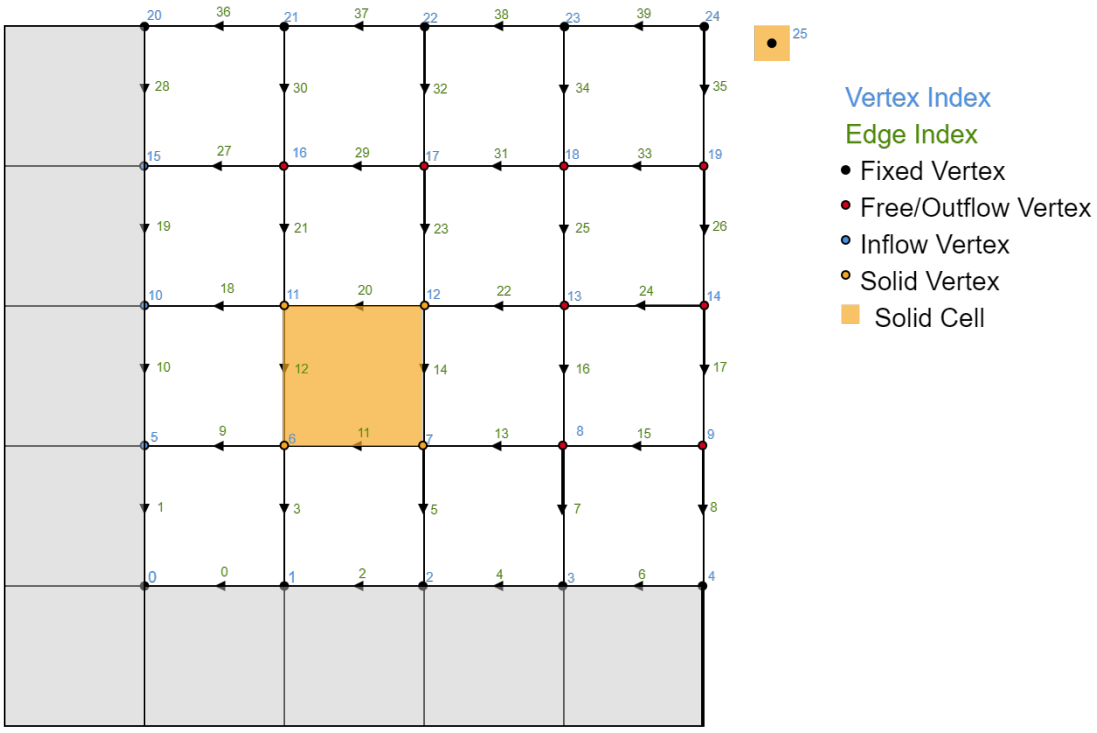


Figure 4.7: *Example Mesh*

	Streamfunction	**Vorticity**
Inflow	$\frac{\partial \Psi}{\partial \tau} = \vec{u}_{in}$	None
Outflow	None	None
Free-Slip	$\frac{\partial \Psi}{\partial \tau} = 0$	$\omega = 0$
No-Slip	$\frac{\partial \Psi}{\partial \tau} = 0, \frac{\partial \Psi}{\partial n} = 0$	$\omega = -\frac{\partial u_\tau}{\partial n}$

Table 4.1: *Boundary Conditions*

4.4.1 Algorithm

Result: Boundary respecting operators

1. create standard operators:
   $\Delta = d_1 d_1^T$;
   $\Delta^2 = \Delta\Delta$;
2. create corrected operators:
   $\Delta_{\text{corrected}} = d_{1c} d_{1c}^T$;
   $\Delta^2_{\text{corrected}} = \Delta_{\text{corrected}}\Delta_{\text{corrected}}$;
3. initialise empty solid matrices:
   $\Delta_{\text{solid}} = \mathbb{0}$;
   $\Delta^2_{\text{solid}} = \mathbb{0}$;
4. reduce solid vertices to solid nodes:

for $i \in \Delta.cols()$ **do**
  if *i.is_solid()* **then**
    $\Delta_{\text{solid}}(i.object\_id()) += i$;
  end
end
for $i \in \Delta^2.cols()$ **do**
  if *i.is_solid()* **then**
    $\Delta^2_{\text{solid}}(i.object\_id()) += i$;
  end
end

5. combine Δ / Δ^2 with Δ_{solid} / Δ^2_{solid}
6. correct fixed rows in Δ_{solid} / Δ^2_{solid}
7. combine $\Delta_{\text{corrected}}$ / $\Delta^2_{\text{corrected}}$ with Δ_{solid} / Δ^2_{solid}
8. set empty diagonal of $\Delta_{\text{corrected}}$ to 1:

for $i \in \Delta_{solid}.diagonal()$ **do**
  if $i = 0$ **then**
    $i = 1$;
  end
end

9. define right-hand side operators:
   $\Delta_{\text{rhs}} = \Delta - \Delta_{\text{corrected}}$;
   $\Delta^2_{\text{rhs}} = \Delta^2 - \Delta^2_{\text{corrected}}$;
10. correct fixed rows in Δ_{rhs} / Δ^2_{rhs}
11. combine biharmonic and Laplacian operators:
    $L = \Delta_{\text{corrected}} + \text{dt}\nu\Delta^2_{\text{corrected}}$;
    $R = \Delta_{\text{rhs}} + \text{dt}\nu\Delta^2_{\text{rhs}}$;

Algorithm 1: Enforcing boundary conditions

In step 1. we create the standard Laplacian and biharmonic matrices as defined in section 4.3.3. Additionally we create the $\Delta_{\text{corrected}}$ and $\Delta^2_{\text{corrected}}$ matrices. These are made by multiplying a modified d_{1c} operator (4.8) with itself. This operator is the same as the d_1 operator (fig. 4.8), except that it does not contain the connections to fixed nodes. This allows us to enforce a fixed streamfunction value on the boundary. Because the streamfunction is fixed on the outer boundaries, we must move these values from the left-hand side of equation 3.60 to the right.

To incorporate solid objects into the equations, we create a separate matrix Δ_{solid}. As each solid object has a constant streamfunction, we represent its streamfunction as one value in the Ψ vector and one row/column in the operators. To populate this Δ_{solid} matrix, we sum up all columns of the Laplacian/biharmonic which correspond to a node contained in the solid object (Alg.1, step 4). Next, we combine the Laplacian and biharmonic operator with their solid counterparts as seen in equations 4.13 and 4.14.

$$\Delta = \left[\begin{array}{c|c} \Delta & \Delta_{\text{solid}} \\ \hline \Delta^T_{\text{solid}} & \mathbb{1} \end{array}\right] \tag{4.13}$$

$$\Delta^2 = \left[\begin{array}{c|c} \Delta^2 & \Delta^2_{\text{solid}} \\ \hline {\Delta^2}^T_{\text{solid}} & \mathbb{1} \end{array}\right] \tag{4.14}$$

In step 6. we then correct the values of solid matrices which correspond to fixed nodes and the combine them with $\Delta_{\text{corrected}}$ and $\Delta^2_{\text{corrected}}$ in the same manner as seen in equations 4.13 and 4.14. To ensure that the operator remains s.p.d. and invertible, we set all values on the diagonal of $\Delta_{\text{corrected}}$ to 1 if they are 0. To compensate for the values which were removed from the left-hand side we define two matrices which contain all the values removed or modified in the original operators:

$$\Delta_{\text{rhs}} = \Delta - \Delta_{\text{corrected}} \tag{4.15}$$

$$\Delta^2_{\text{rhs}} = \Delta^2 - \Delta^2_{\text{corrected}} \tag{4.16}$$

In a final step we combine the Laplacian and biharmonic operators.

$$\mathbf{L} = \text{dt}\nu\Delta^2_{\text{corrected}} - \Delta_{\text{corrected}} \tag{4.17}$$

$$\mathbf{R} = \text{dt}\nu\Delta^2_{\text{rhs}} - \Delta_{\text{rhs}} \tag{4.18}$$

The linear system of equations that we now solve is given by

$$\mathbf{L}\vec{\Psi}^{n+1} = \omega^* + \mathbf{R}\vec{\Psi}^{\text{fixed}} \tag{4.19}$$

The matrices below represent the mesh seen in figure 4.7. The d_1 operators are (25×40)-Matrices, where the rows represent vertices and the columns represent edges in the mesh. The laplacian and biharmonic operators are all (26×26)-Matrices. Both the rows and columns represent vertices. The final row/column represent the solid cell in the fluid domain.

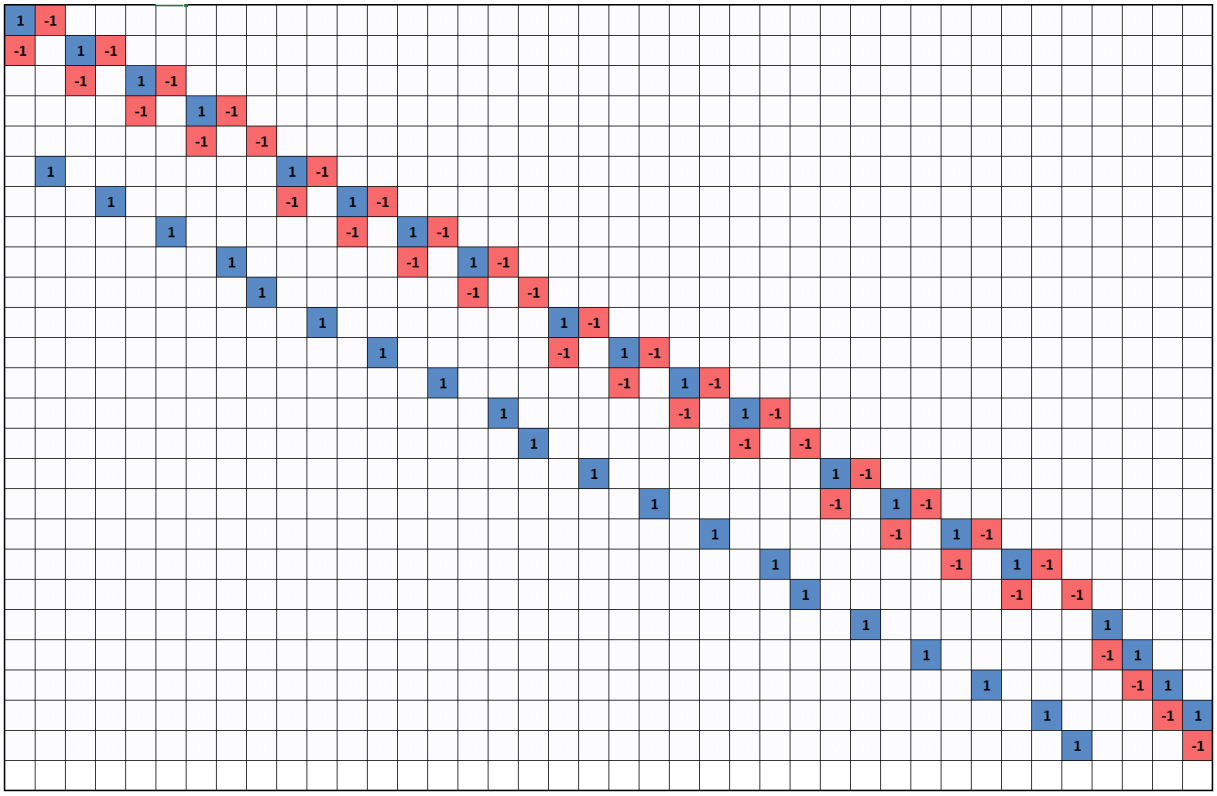


Figure 4.8: *Delta 1 operator*

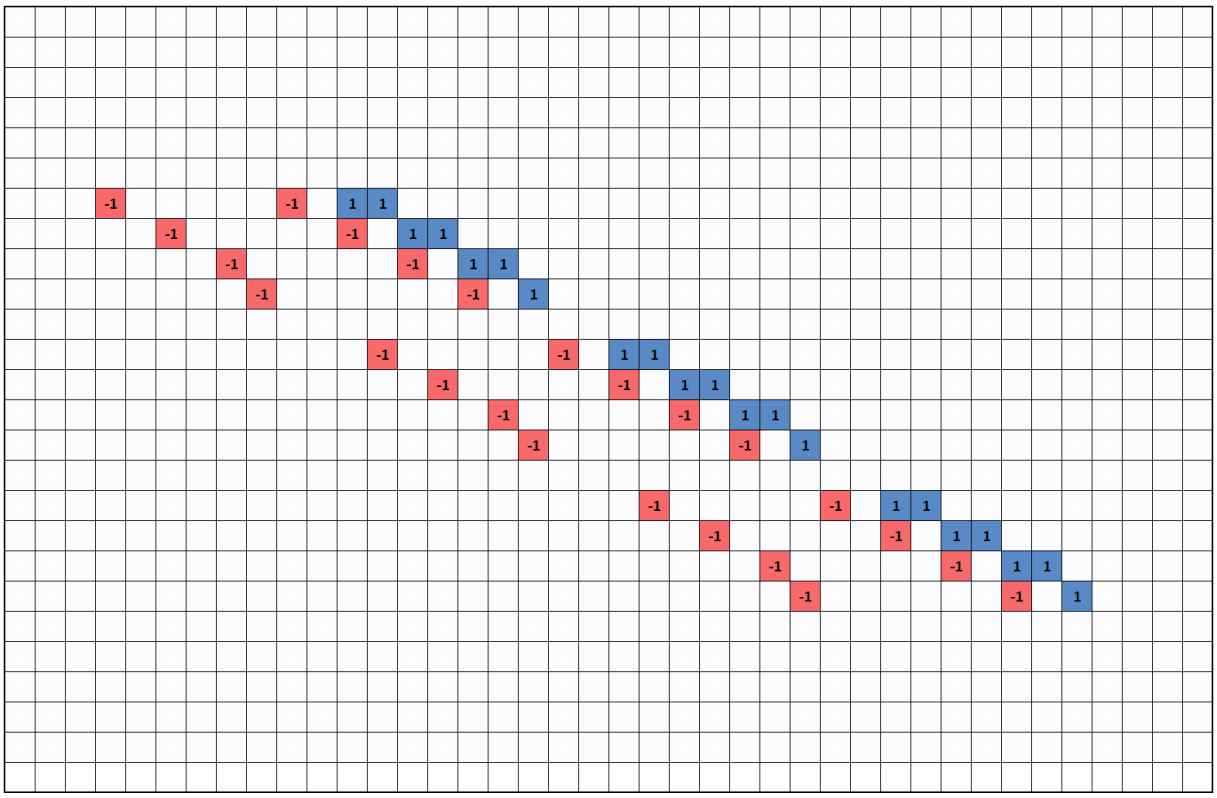


Figure 4.9: *Corrected Delta 1 pperator*

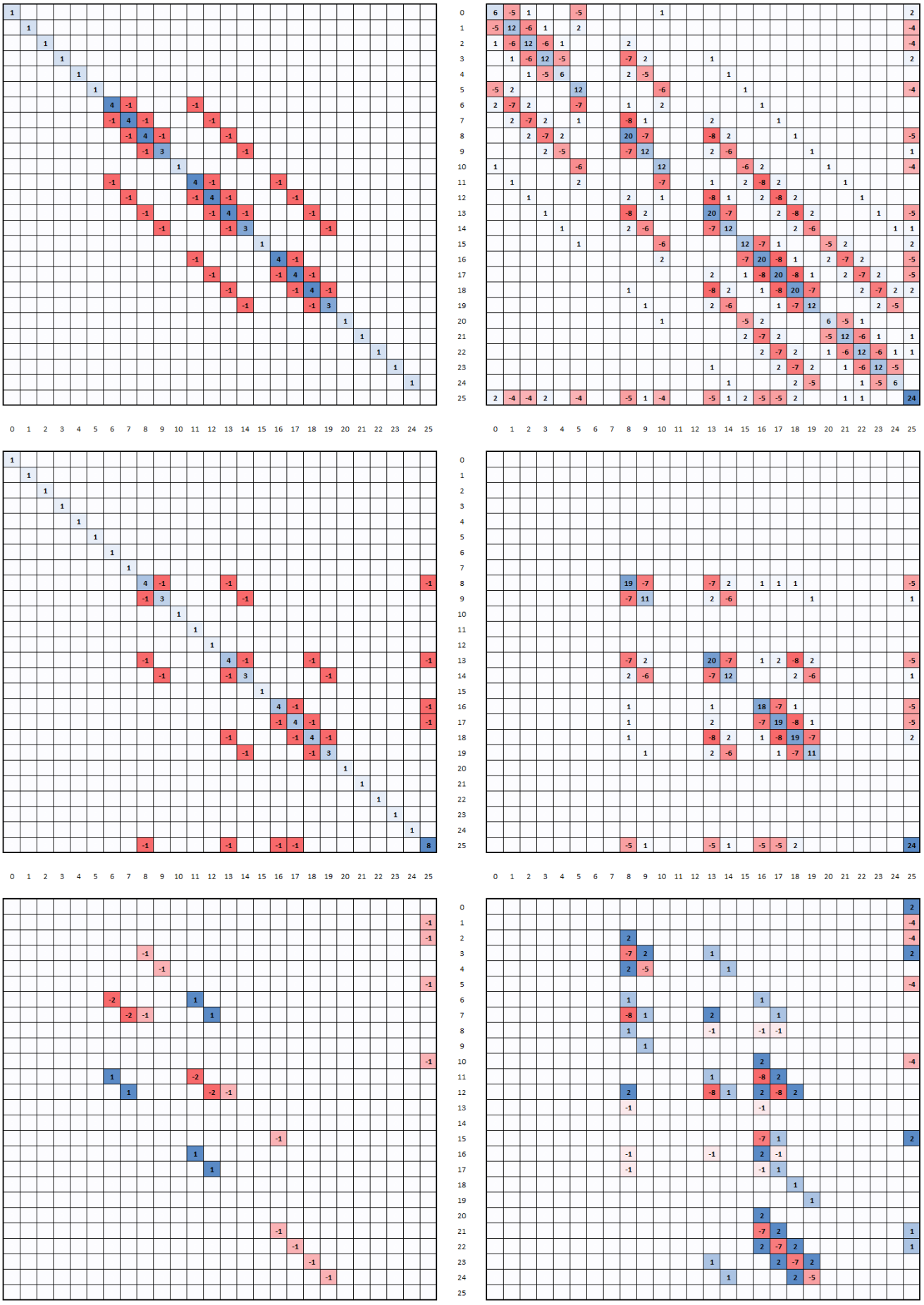


Figure 4.10: *Left column: Laplacian, right column: biharmonic operator; top row: original, middle row: corrected, bottom row: right-hand side*

Results

5.1 Configurations

In the following section we will compare our solver with a pressure-velocity and other similar solvers. We will show how the solver performs handling different boundary conditions, solid objects and diffusion. All simulations were performed under the following parameters:

Parameter	Value
dx	0.0625
dt	0.0078
FLIP/PIC ratio	0.001

Table 5.1: *Parameters used in all simulations*

5.2 Examples

5.2.1 Empty Channel

In this example we simulate an empty channel of dimensions 8×3. The western boundary is constrained with an inflow rate of 3. The southern and northern boundaries are solid. Simulations A and B have free-slip boundary conditions and simulations C and D have no-slip boundaries. The eastern boundary is an outflow boundary. Simulations A and C use our streamfunction-vorticity method. Simulations B and D use a pressure-velocity solver.

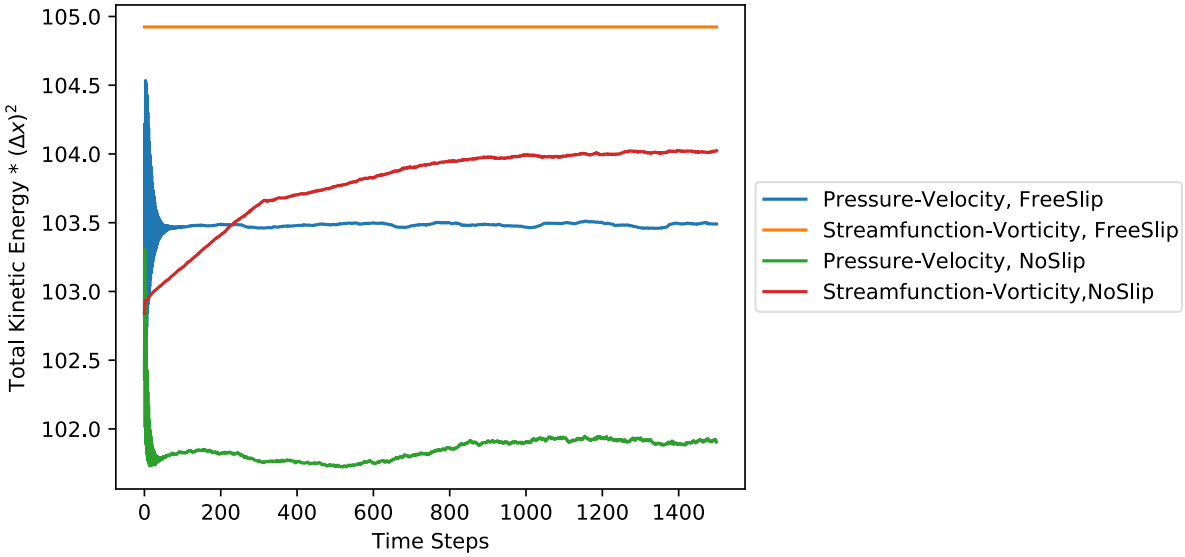


Figure 5.1: *Kinetic energy, empty channel*

5.2.2 Object in Channel

As with the example above, we have a channel with the same dimensions and boundary conditions. The channel contains a circle of radius 0.4 with its center at (2,1.5).

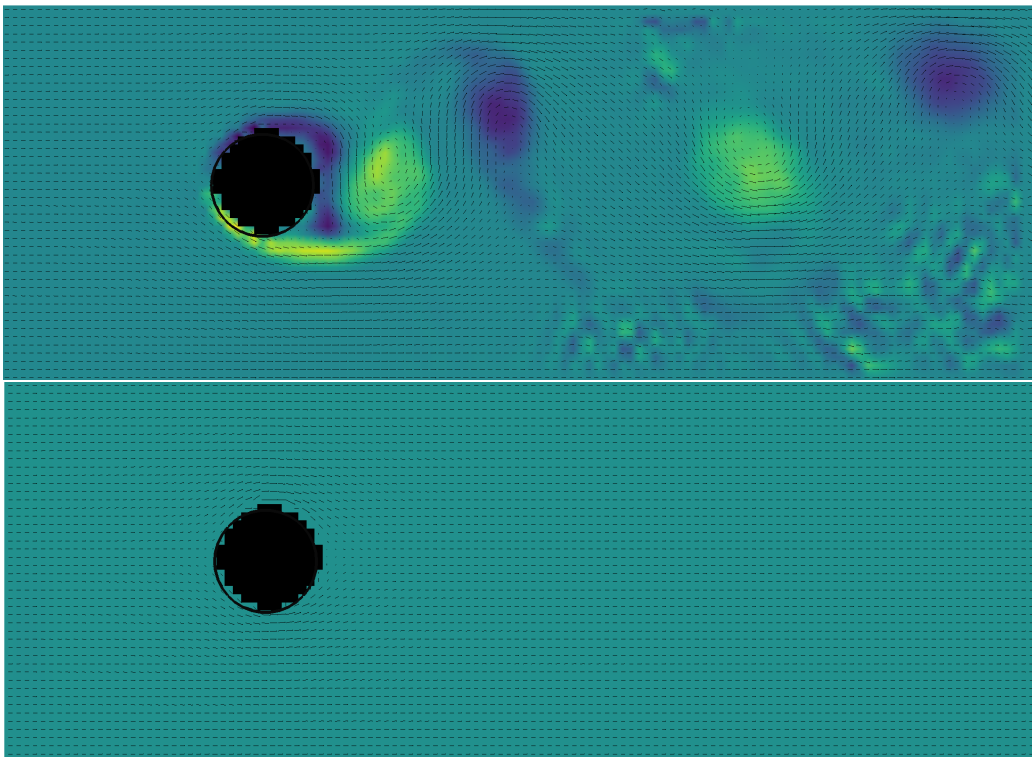


Figure 5.2: *FreeSlip, top: pressure-velocity, bottom: streamfunction-vorticity*

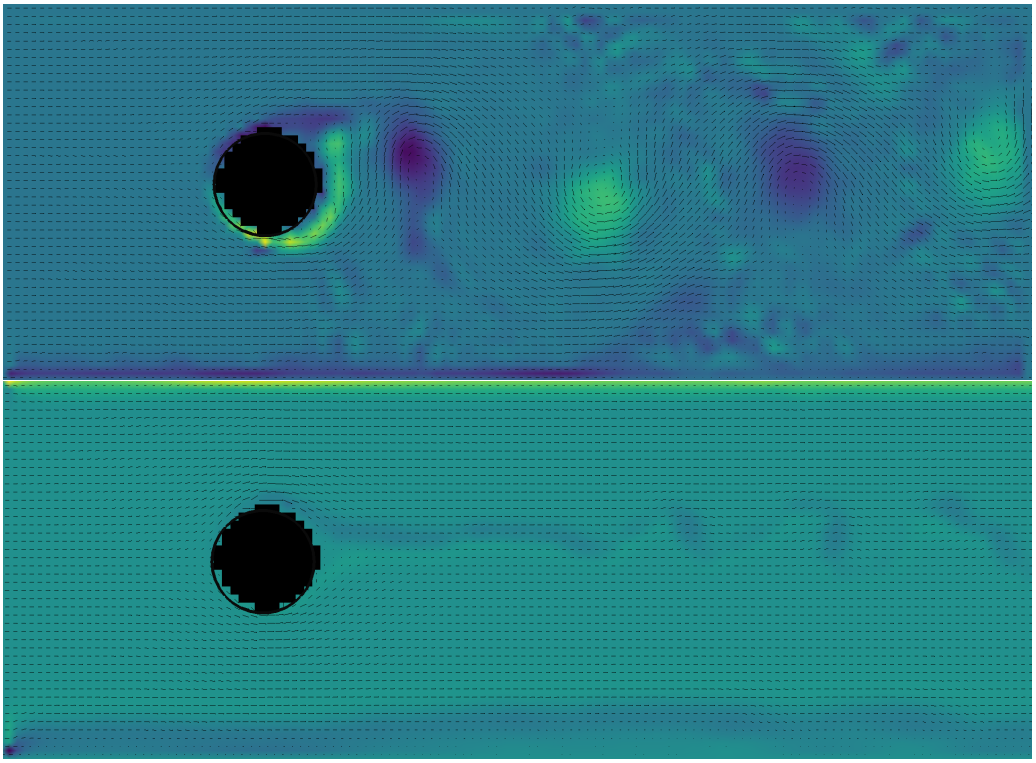


Figure 5.3: *NoSlip, top: pressure-velocity, bottom: streamfunction-vorticity*

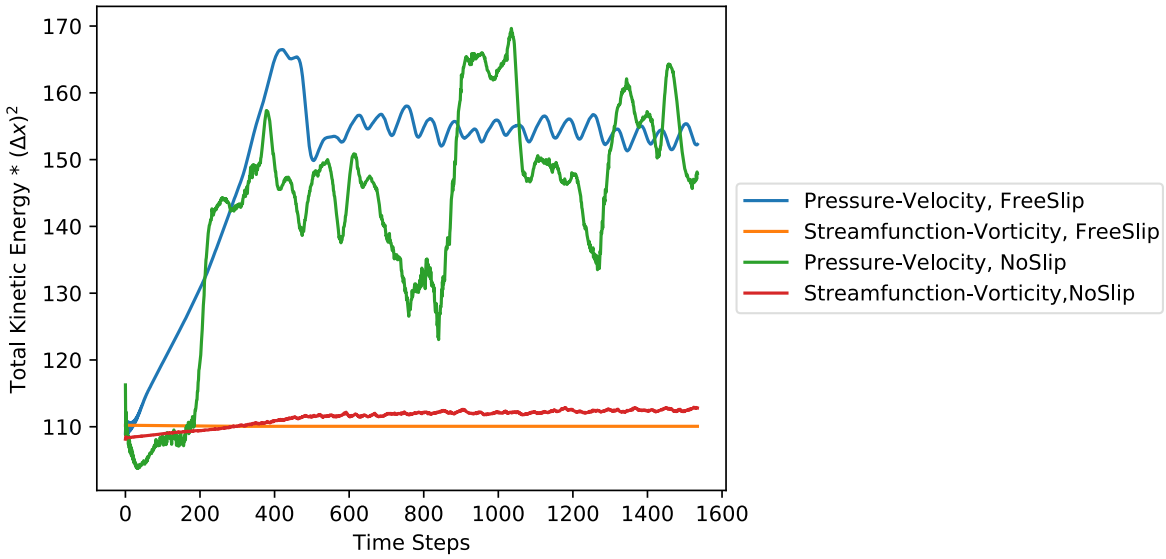


Figure 5.4: *Kinetic energy, object in channel*

5.2.3 Object in Channel with diffusion

In this example we examine the same object channel configuration as the previous example but incorporate different diffusion coefficients. All simulations use our streamfunction-vorticity solver, an 8×3 simulation domain and circle of radius 0.4 with its center at (2,1.5).

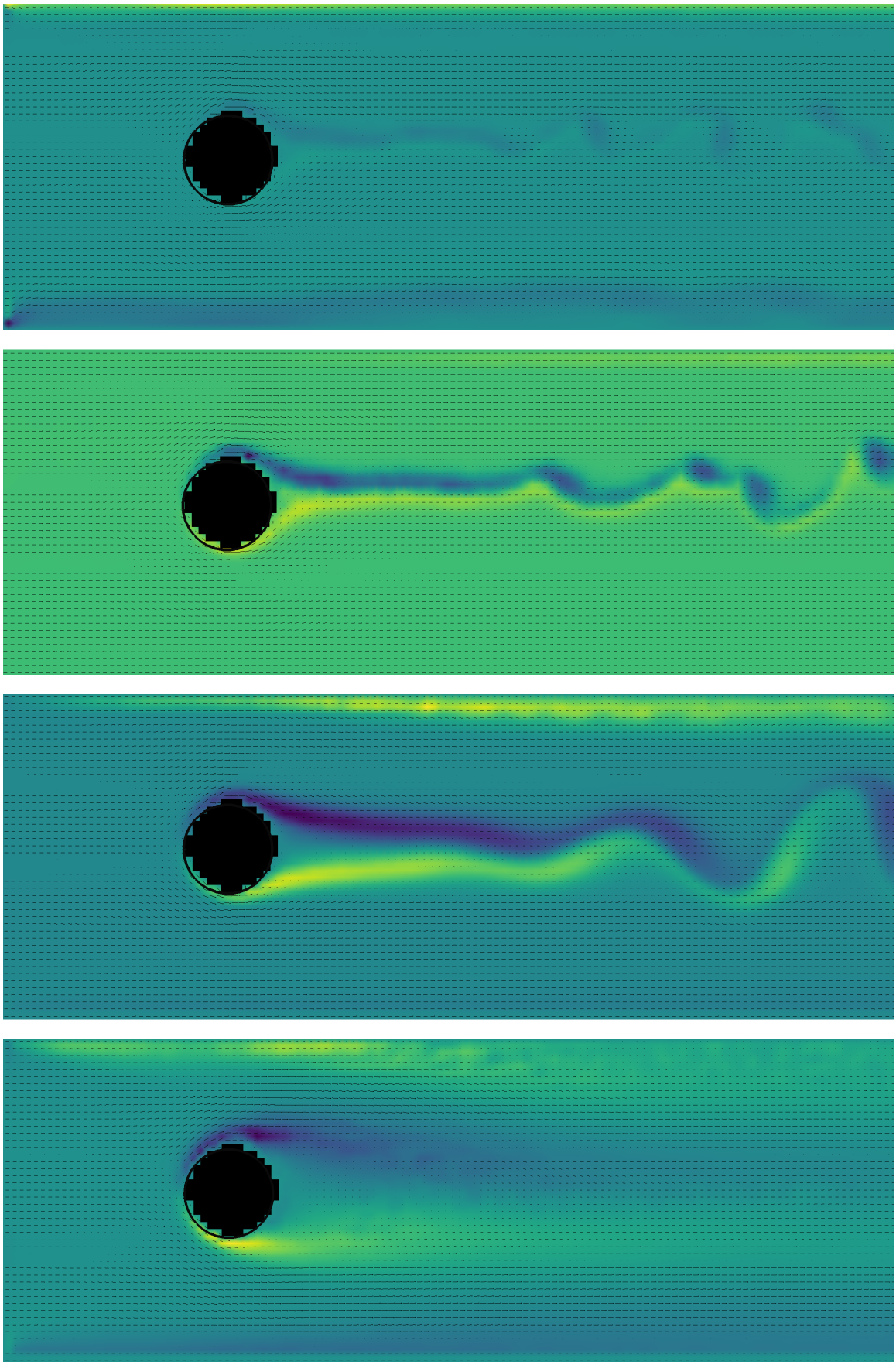


Figure 5.5: *Object in channel with diffusion, 1st row:* $\nu = 0$*, 2nd row:* $\nu = 0.001$*: 3rd row:* $\nu = 0.01$*, 4th row:* $\nu = 0.1$

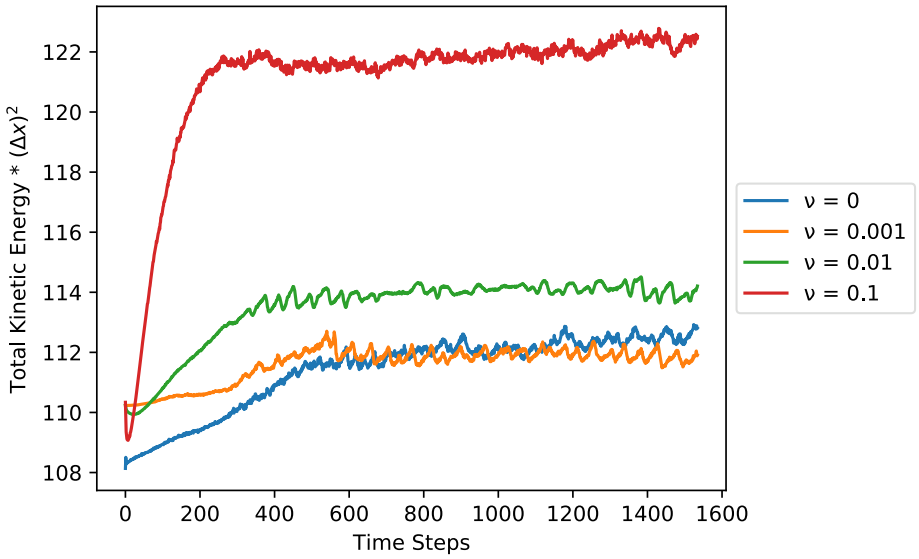


Figure 5.6: *Kinetic energy, object in channel with diffusion*

5.2.4 Taylor Vortex

In this example we simulate a domain which is initialized with two vortices as described in [McK07] eqn. 1.16. The initial vorticity is also depicted in figure 5.7. The domain is $(-\pi, \pi) \times (-\pi, \pi)$ and is discretized with a 128×128 grid. All boundaries are free-slip and the domain contains no solid objects.

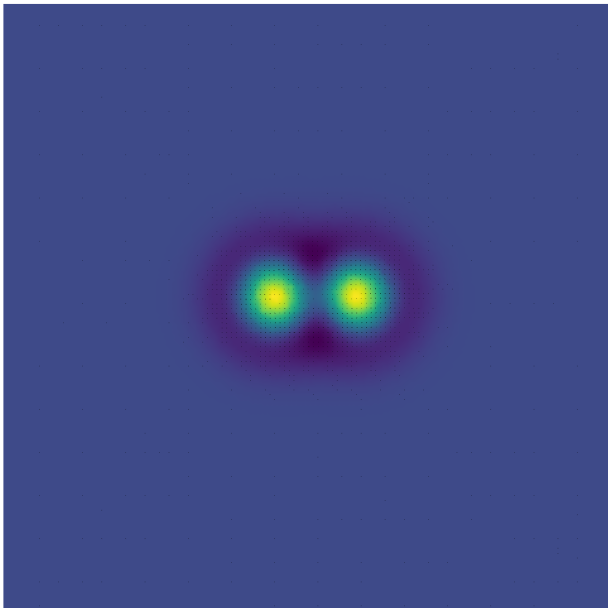


Figure 5.7: *Initial vorticity for all simulations*

The vorticity is initialized with with a vortices at $(-0.4, 0)$ and $(0.4, 0)$.

$$\omega(x,y) = \frac{U}{a}\left(2 - \frac{r^2}{a^2}\right)\exp\left(\frac{1}{2}\left(1 - \frac{r^2}{a^2}\right)\right) \tag{5.1}$$

With r being the distance from point (x, y) to the center of the vortex, a is the core size of the vortex and U is the maximum tangential velocity. We ran the simulations with $a = 0.3$ and $U = 1$ for both vortices. The vorticity at each grid point is the sum of values the vortex at the given grid point.

We ran the simulation 4 times with different diffusion coefficients.

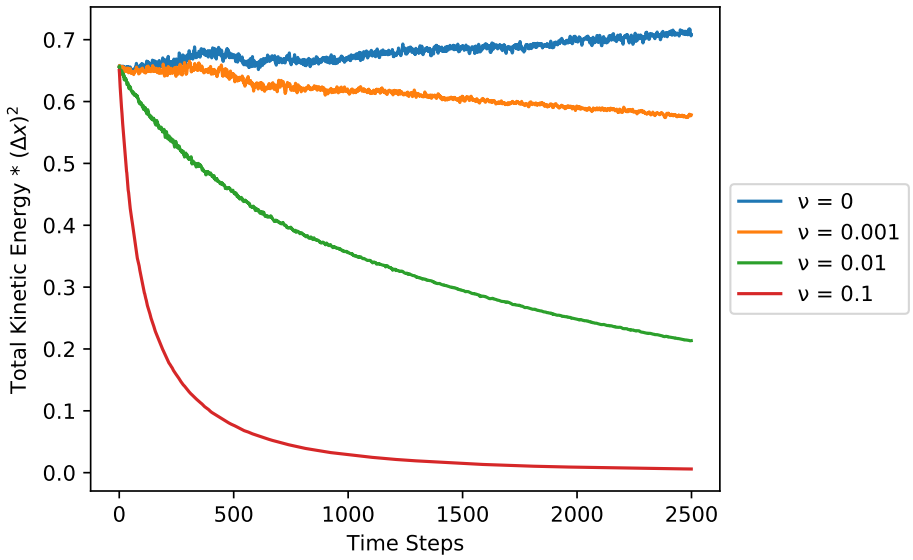


Figure 5.8: *Kinetic energy, taylor vortex*

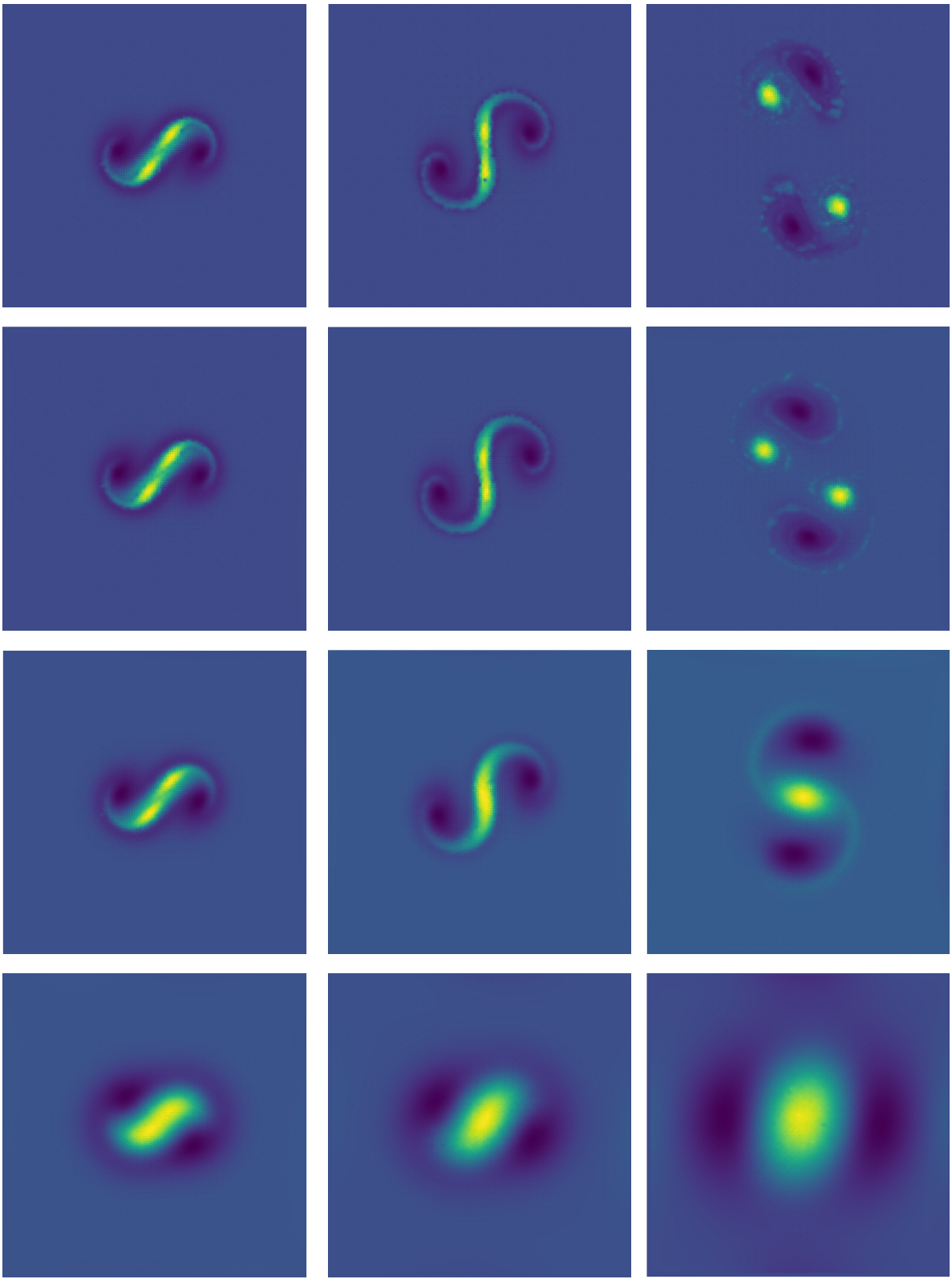


Figure 5.9: *Taylor vortex: 1st row:* $\nu = 0$*, 2nd row:* $\nu = 0.001$*: 3rd row:* $\nu = 0.01$*, 4th row:* $\nu = 0.1$

Conclusion

6.1 Conclusion

In our work we were able to implement an efficient streamfunction vorticity solver. By discretizing the Navier-Stokes equations with discrete exterior calculus, we built a flexible and robust numerical framework which facilitates the possibility of extending the solver to non-uniform meshes. By using the streamfunction vorticity formulation of the Navier-Stokes equations, the solution space of our flow only incorporates divergence-free flows. This allows us to eliminate the diffusive projection step needed in pressure-velocity simulations. Furthermore, we incorporate a parameterized viscosity term which modifies the Laplacian operator and enables the simulation of different viscid and inviscid fluids.

In sections 5.2.1 and 5.2.2 we see that our method can produce results with very little numerical viscosity. This is shown by the extremely stable kinetic energy of our free slip simulations. In figure 5.2 we see that this constant energy comes from our ability to simulate laminar flows. Due to the numerical viscosity of the pressure-velocity method, it is only able to depict turbulent flows. In figures 5.4 and 5.3 we see that by adding no slip boundaries we are able to create small vortices with our object submerged in the flow. These vortices are well preserved in the simulation due to the lack of viscosity.

In figure 5.5 we show our ability to modify the fluid characteristics by changing the Reynolds number through our viscosity parameter ν. With $\nu = 0$ and no slip conditions we have a stable flow with minor turbulence. By increasing the viscosity, we can increase the turbulence, create von Kármán vortex streets and finally achieve flow separation. The Reynolds number in the simulations are approximately $Re > 10^4, Re \sim 10^3, Re \sim 10^2, Re \sim 10^2$ respectively. According to [TM01] vortex streets occur for Reynolds numbers in the range $40 < Re < 10^3$. The results from our simulations correspond well with these assumptions.

Due to errors in the interpolation scheme, no-slip boundaries have different effect on the top

and bottom boundaries. This cause the slightly asymmetric flow seen in figure 5.5. For higher viscosity fluids, this can lead to more vortex shedding and flow separation on the upper boundary.

The versatility of our solver is also well illustrated in our simulation of a Taylor-vortex example in section 5.2.4. In figure 5.9 we see that low viscosity simulations ($\nu = 0$ & $\nu = 0.001$) are able to retain the two separate vortices and preserve their energy over a long period of time. By increasing the viscosity, we are able merge the two vortices and smoothly remove energy from the system.

In section 1.4.2 of [McK07] we see how a variety of other solvers simulate this problem. In his thesis, McKenzie compares his High-Order Lie Advection (HOLA) solver with a pseudospectral method [HZ87] and circulation preserving scheme [ETK$^+$07]. When comparing our simulations in figure 5.9 to figure 1.6 of McKenzie's thesis [McK07], it becomes clear that our method can mimic a variety of different higher and lower order methods.

Our solver is versatile and offers fine grain control of viscosity from highly inviscid to viscid fluids. Due to this control, we can offer both energy-preserving and dissipative simulations. Because the solution space of our fluid flow is limited to incompressible, divergence-free fields, our solver can be very sensitive to boundary conditions. It also requires sufficient iterations of the conjugate gradient solver of the streamfunction equation (4.19) to remain stable. Because of the non-dissipative nature of our solver, errors in this computation can build up over time and lead to unstable solutions.

6.2 Future Research

Our proposed solver is only able to simulate domains which are completely filled with a fluid. This means it is not yet possible to model the interaction between water and air. Future work could implement our viscosity term of the Navier-stokes equations into the cut-cells solver created by Safira Piasko [Pia20]. One could then define the necessary surface boundary conditions and map these on a cut-cell boundary. Due to the flexibility of our discrete exterior calculus framework, it is possible to efficiently adapt the mesh to incorporate complex boundaries. Another important addition to our work would be to extend our solver into three-dimensions. Whilst all the mathematical foundations of our work are valid in three dimensions, it is necessary to efficiently compute the linear system of equations. This can be done with help from the null-space reduction techniques proposed by Livio Steiner [Ste18]. Finally, alternatives to FLIP/PIC based advection may be advisable. Problems with the diffusive characteristics of PIC and particle-to-grid interpolation could be improved and the stability of the simulation could be improved by removing FLIP advection.

Bibliography

[AN05] Alexis Angelidis and Fabrice Neyret. Simulation of Smoke Based on Vortex Filament Primitives. In Ken Anjyo and Petros Faloutsos, editors, *Symposium on Computer Animation (SCA '05)*, pages 35–48, Los Angeles, United States, July 2005. ACM-SIGGRAPH/EG, ACM Press.

[BKR88] J.U. Brackbill, D.B. Kothe, and H.M. Ruppel. Flip: A low-dissipation, particle-in-cell method for fluid flow. *Computer Physics Communications*, 48(1):25–38, 1988.

[Cal02] Donna Calhoun. A cartesian grid method for solving the two-dimensional streamfunction-vorticity equations in irregular regions. *Journal of Computational Physics*, 176, 2002.

[DG96] Mathieu Desbrun and Marie-Paule Gascuel. Smoothed particles: A new paradigm for animating highly deformable bodies. In Ronan Boulic and Gerard Hégron, editors, *Computer Animation and Simulation '96*, pages 61–76, Vienna, 1996. Springer Vienna.

[ETK+07] Sharif Elcott, Yiying Tong, Eva Kanso, Peter Schröder, and Mathieu Desbrun. Stable, circulation-preserving, simplicial fluids. *ACM Trans. Graph.*, 26(1):4–es, January 2007.

[FM96] Nick Foster and Dimitri Metaxas. Realistic animation of liquids. *Graphical Models and Image Processing*, 58(5):471–483, 1996.

[Fro64] J Fromm. Methods in computational physics, 1964.

[GJ+10] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[Har] Francis H Harlow. The particle-in-cell method for numerical solution of problems in fluid dynamics.

[HS+52] Magnus R Hestenes, Eduard Stiefel, et al. Methods of conjugate gradients for

solving linear systems. *Journal of research of the National Bureau of Standards*, 49(6):409–436, 1952.

[HZ87] M Y Hussaini and T A Zang. Spectral methods in fluid dynamics. *Annual Review of Fluid Mechanics*, 19(1):339–367, 1987.

[KC13] Mathieu Desbrun Peter Schröder Keenan Crane, Fernando de Goes. Digital geometry processing with discrete exterior calculus. In *ACM SIGGRAPH 2013 courses*, SIGGRAPH '13, New York, NY, USA, 2013. ACM.

[McK07] Alexander McKenzie. Hola: a high-order lie advection of discrete differential forms. Master's thesis, 2007.

[PDV] Chris Buck Walt Disney Studios Motion Pictures Peter Del Vecho, Jennifer Lee. Frozen 2.

[Pia20] Safira Piasko. On reducing numer-ical dissipation of euler equations with discrete exterior calculus, 2020.

[Sam18] Daniel Valério Sampaio. An efficient boundary-respectingstreamfunction-vorticity solver. Master's thesis, 2018.

[Ste18] Livio Steiner. Reducing numerical dissipation of euler equations for computer graphics. Master's thesis, 2018.

[Sto45] G. G. Stokes. On the theories of the internal friction of fluids in motion and of the equilibrium and motion of elastic solids. *Transactions of the Cambridge Philosophical Society*, Vol. 8, 1845.

[TM01] C. E. Tansley and D. Marshall. Flow past a cylinder on a plane, with application to gulf stream separation and the antarctic circumpolar current. *Journal of Physical Oceanography*, 31:3274–3283, 2001.

[ZB05] Yongning Zhu and Robert Bridson. Animating sand as a fluid. *ACM Trans. Graph.*, 24(3):965–972, July 2005.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

An Efficient Streamfunction-Vorticity Solver for the Simulation of Viscous Fluids

Authored by (in block letters):

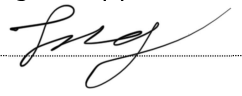
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Lausberg	Tom

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich 30.04	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.