
MASTER THESIS

---

# Two-dimensional Characteristic Mapping Method with inertial particles on GPU using CUDA

---

at

Institut de Mathématiques de Marseille

by

Nicolas SABER

Supervisors

Prof. Kai Schneider

Thibault Oujia

September 17, 2021

Master 2 Mathématiques appliquées
Calcul scientifique, Équations aux dérivées partielles, Probabilités, Statistique

**Abstract**

The numerical resolution of the 2D incompressible Euler equations is challenging due to the nonlinear transport, sudden steepening of gradients and the missing regularization by viscosity. Here we investigate the Characteristic Mapping Method (CMM) which is a semi-Lagrangian method with high precision on a coarse grid. The flow map is evolved using a gradient augmented level set method and FFT techniques allow efficient reconstruction of the velocity field. Remapping allows error control of the incompressibility condition. Computational efficiency and the high precision of CMM are analyzed optimizing an existing cuda code, which is implemented and adapted on recent graphic cards. Different validation tests are performed illustrating third order accuracy. The transport of inertial and fluid point particles is likewise implemented and the influence of inertia is studied.

**Acknowledgements**

This project would not have been possible without the support of many people. I would like to thank my supervisors Prof. Kai Schneider and Thibault Oujia from I2M (Aix Marseille University) for their help into guiding my work, their precious advice and also for reading the numerous revisions of this master thesis. I would also like to thanks Xi-Yuan Yin (McGill University, Montreal) for his constructive comment and for providing me the exact parameters used for the validation and Julius Bergmann (TU Berlin, now at I2M) for his revision and for setting the github page where the code will be hosted and accessible for the community. Centre de Calcul Intensif d'Aix-Marseille is acknowledged for granting access to its high performance computing resources and to the new graphics card machine of I2M, called Anticythere. Financial support from I2M, the ALEA group and the ANR project CM2E is thankfully acknowledged.

# Table of contents


**1 Introduction** **1**

**2 2D incompressible Euler equations** **3**
2.1 Properties of the Navier-Stokes and Euler equations . . . 4
2.1.1 Balance equations . . . 4
2.1.2 Isotropic energy spectrum . . . 5
2.2 Flow map . . . 7

**3 The Characteristic Mapping Method** **8**
3.1 Formal aspect . . . 8
3.1.1 Framework . . . 8
3.1.2 Semi-group structure . . . 9
3.2 Numerical scheme and implementation . . . 10
3.2.1 Remapping . . . 10
3.2.2 Grids and Multiscale Representation . . . 11
3.2.3 Lagrange interpolation of the velocity . . . 11
3.2.4 Time discretization . . . 13
3.2.5 Computation of derivatives through FFT . . . 15

**4 Fluid particles and inertial particles** **15**
4.1 Fluid Particles . . . 15
4.2 Inertial Particle Advection . . . 18

**5 Numericals Results** **21**
5.1 Studied flows . . . 21
5.1.1 Single shear layer . . . 21
5.1.2 4-mode flow . . . 22
5.1.3 Two-dimensional turbulence . . . 24
5.1.4 Two co-rotating vortices . . . 25
5.2 Code validation . . . 26
5.2.1 Time evolution of energy, enstrophy and palinstrophy . . . 26
5.2.2 Enstrophy spectrum . . . 26
5.2.3 Flow visualization . . . 28
5.2.4 CPU/GPU time comparison . . . 30
5.3 Convergence tests . . . 32

**6 Conclusions and perspectives** **35**

**7 Appendix** **37**
7.1 Hermite interpolation in 2D . . . 37
7.2 Flowchart . . . 38

## Notation

In this section, we define some notation and give a list of symbols used throughout this thesis in two dimensions :
Vector quantities are represented by bold symbols and represent different objects :

$\mathbb{T}^2 = \mathbb{R}^2/\mathbb{Z}^2 \subset \mathbb{R}^2$ is a 2D flat-torus in the plane

$\boldsymbol{x} = (x, y) \in \mathbb{R}^2$ is a position in the plane

$\boldsymbol{u} = (u_1, u_2) \in \mathbb{R}^2$ is used for the velocity of the fluid.

$\boldsymbol{v} = (v_1, v_2) \in \mathbb{R}^2$ represents the particle velocity

$\boldsymbol{X} : \mathbb{T}^2 \times \mathbb{R}_+ \to \mathbb{T}^2$ is the formal backward map

$\boldsymbol{\chi} \in \mathcal{M}_n(\mathbb{R})$ represents the numerical characteristic and $n^2 \in \mathbb{N}$ its resolution.

$\nabla f = gradf$ ($f$ is scalar valued)

$\nabla \cdot \boldsymbol{u} = div(\boldsymbol{u})$

$\nabla \times \boldsymbol{u} = curl(\boldsymbol{u})$

$\nabla^2 = \Delta$ is the Laplace operator.

$\nabla^\perp = (-\partial_y, \partial_x)$ is the perpendicular gradient

$\nu$ is the kinematic viscosity

$p : \mathbb{T}^2 \times \mathbb{R}_+$ represents the pressure of the fluid

$\rho \in \mathbb{R}_+$ is the density of the fluid

# 1 Introduction

The different systems of equations governing incompressible fluids, such as the Euler equations and the Navier-Stokes equations have fascinated generations of mathematicians, notably due to the phenomenon called turbulence which appears in high or infinite Reynolds number systems, or in other words, with small or even vanishing viscosity, respectively. The Reynolds number measures the ratio of the strength of the nonlinear term and the linear term and is inversely proportional to viscosity. This phenomenon complicates answers about the explicit solution of these nonlinear PDEs, its stability and its well-posedness. Global existence and uniqueness of the solution of Euler and Navier-Stokes in the 3D case is still an open problem, whereas for the 2D case, we have global existence and uniqueness, proven 1963 by Youdovitch. For the review on the state of the art we refer to Bardos and Titi [2013]. The numerical aspect which is crucial in many scientific and industrial applications, focuses on algorithms and techniques that allow big flow simulations to be run, indeed one of the problems faced in numerical flow studies is that the number of degrees of freedom or grids points increases with $Re$ in 2D and with $Re^{9/4}$ in 3D. This shows that high Reynolds number flow require expensive computations and it is particularly true in the case of the Euler equations which do not have any cut-off scale.

Some of these algorithms could benefit from the progress in technology of the last decade, i.e. GPU parallelization, which is now widely used in the domain of AI (artificial intelligence) training and can potentially open a lot of possibilities for heavy computations when it comes to reduction of computation time. Currently, the biggest computations using FFT based spectral methods for Navier-Stokes have a resolution of $12288^3$ grid points, see e.g. for a discussion and overview on the state of the art [Yin et al., 2021b]. The purpose of this internship is to work on the Characteristic Mapping Method (CMM) for the two-dimensional Euler equations [Yin et al., 2021a]. The starting point is an already existing implementation of CMM, developed by Yadav [2015]. This new numerical method for transporting arbitrary sets in a velocity field uses a deformation mapping of the domain of definition. The implementation for the 2D Euler equation is written in CUDA/C++ which is an API (Application programming interface) working on top of C++ allowing us to utilize graphics cards and its thousands of cores to parallelize the needed procedures in order to improve the overall computation time.

One goal of the master project is to upgrade and improve the existing code and make it more readable. Moreover, it should be updated to work with the last version of the CUDA toolkit on the new machine equipped with a NVIDIA TESLA V100S at I2M. Furthermore, there are several numerical questions and aspects regarding the implementation, such as changing the time integration from a linear multistep method Adams-Bashforth to a one step Runge-Kutta method. To this end, Lagrange time interpolation is proposed to provide intermediate time steps. Furthermore the evolution of inertial point particles based on the Maxey-Riley model is implemented considering different values for the particle inertia, i.e. light to heavy particles. Altogether 21 simulations with different Stokes numbers were carried out in order to further test and validate the CM method. These particles will be used to better study turbulent flow and its singularities. The code will also be subject of benchmarking tests covering different aspects : CPU/GPU time, convergence properties, memory usage.

The master thesis is organised as follows : Firstly, we recall in section two the incompressible Euler equations, the vorticity equation and the Biot-Savart law for the velocity field. In addition, we will introduce an advection/Biot-Savart coupling as well as different characteristic flow quantities e.g. kinetic energy, enstrophy, which are conserved quantities in the 2D Euler equations.

Afterwards in section three, we introduce the essential parts of the CM method such as the

flow map and its properties. We will describe the numerical questions and aspects of implementation, such as the remapping process, the grids used in order to save the small/large scale features of the flow and the spatial/time discretization used in the code.

The fourth section is reserved for point particles advected by the fluid flow. We firstly transport fluid particles, i.e. particles without mass. This is an extension of the fluid simulations since point particles just follow along the fluid flow. Following, we will focus in more detail on inertial point particles and the numerical implementation of the Maxey-Riley model. This has the goal to better understand flow turbulence and clustering by studying the behaviour of the particles before loosing the ability to reverse the simulation.

The fifth section is devoted to the code validation by comparing the present results with reference simulations presented in Yin et al. [2021a] and trying to match the properties of the MATLAB implementation [Yin et al., 2021a]. We study enstrophy spectra, the flow visualization as well as the convergence errors of different quantities and their relations towards the time-step and the spatial-step against the reference simulation. This fifths section ends with results for the particles and turbulence, while also speaking about the influence of the inertia of the particles and some benchmarking for the time spent.

Finally, in section 6 conclusions of the present work are drawn and some perspectives for future work are given.

## 2 2D incompressible Euler equations

The flow of an incompressible viscous fluid is governed by the Navier-Stokes equations

$$\begin{cases} \partial_t \boldsymbol{u} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} &= -\dfrac{1}{\rho} \nabla p + \nu \Delta \boldsymbol{u} \quad \text{in } \mathbb{T}^d \times \mathbb{R}_+ \\ \nabla \cdot \boldsymbol{u} &= 0 \\ \boldsymbol{u}(\boldsymbol{x}, 0) &= u_0(\boldsymbol{x}) \end{cases} \tag{1}$$

where $\boldsymbol{u}_0, \boldsymbol{u}$ and $p$ have appropriate periodic boundary conditions on $\mathbb{T}^d = \mathbb{R}^d/\mathbb{Z}^d$ which is a $d$-flat Torus and can be taken as a $[0, 2\pi]^d$ square (homeomorphism). Here $\boldsymbol{u}$ represents the velocity field, $p$ is the pressure field, $\nu$ is the kinematic viscosity and $\rho$ the density which we assume to be constant. Here we take $\rho = 1$.

The two partial differential equations in eq. (1) represent the conservation of momentum and mass. The incompressible Euler equations are then obtained from the Navier-Stokes equations setting $\nu = 0$. They describe the flow of an incompressible perfect fluid :

$$\begin{cases} \partial_t \boldsymbol{u} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} &= -\nabla p \quad \text{in } \mathbb{T}^d \times \mathbb{R}_+ \\ \nabla \cdot u &= 0 \\ \boldsymbol{u}(\boldsymbol{x}, 0) &= \boldsymbol{u}_0(\boldsymbol{x}) \end{cases} \tag{2}$$

Introducing the vorticity, $\boldsymbol{\omega} = \nabla \times \boldsymbol{u}$ which represents the local rotation of the fluid, we obtain a nonlinear transport equation for $\boldsymbol{\omega}$ by taking the curl of (2):

$$\begin{cases} \partial_t \boldsymbol{\omega} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{\omega} &= (\boldsymbol{\omega} \cdot \nabla)\boldsymbol{u} \\ \nabla \cdot \boldsymbol{u} &= 0 \\ \boldsymbol{\omega}(x, 0) &= \boldsymbol{\omega}_0(x) \end{cases} \tag{3}$$

Considering two-dimensional flows ($d = 2$), i.e. flows in the plane, we only have two velocity components and each component depends on two directions $\boldsymbol{u}(\boldsymbol{x}, t) = (u_1(\boldsymbol{x}, t), u_2(\boldsymbol{x}, t), 0)$. The 3D vorticity vector then only has one non vanishing component $\boldsymbol{\omega}(\boldsymbol{x}, t) = (0, 0, \omega_3(\boldsymbol{x}, t))$ and vorticity becomes a pseudo-scalar being reduced to $\omega = \omega_3(\boldsymbol{x}, t) = \partial_x u_2 - \partial_y u_1$. Furthermore, $\omega$ being perpendicular to $\boldsymbol{u}$, the vortex stretching term $\omega \cdot \nabla \boldsymbol{u}$ in (3) vanishes, thus obtaining a transport equation for the vorticity $\omega$ in the plane :

$$\begin{cases} \partial_t \omega + (\boldsymbol{u} \cdot \nabla)\omega &= 0 \\ \nabla \cdot \boldsymbol{u} &= 0 \end{cases} \tag{4}$$

The incompressibility of the velocity field, $\nabla \cdot \boldsymbol{u} = 0$, implies from the Poincaré lemma the existence of a stream function, $\psi : \mathbb{T}^2 \times \mathbb{R}_+$ with:

$$\nabla^\perp \psi = \boldsymbol{u} \tag{5}$$

and where $\nabla^\perp$ is defined as the rotated (or perpendicular) gradient, i.e. :

$$\nabla^\perp = (-\partial_y, \partial_x)$$

Expressing $\omega$ by taking the curl of (5), we obtain :

$$\omega = \nabla \times (\nabla^\perp \psi)$$

By expanding the right hand side :

$$\begin{aligned}\omega &= \nabla \times (\nabla^{\perp}\psi) \\ &= \nabla \times (-\partial_y\psi, \partial_x\psi) \\ \omega &= \partial_x^2\psi + \partial_y^2\psi\end{aligned}$$

Hence, we obtain a Poisson equation for the stream function:

$$\omega = \nabla^2\psi = \Delta\psi \tag{6}$$

Finally, by combining equation (5) and 6 we can deduce the velocity $\boldsymbol{u}$ using the Biot-Savart kernel:

$$\boldsymbol{u} = \nabla^{\perp}\Delta^{-1}\omega \tag{7}$$

where $\Delta^{-1}$ is the inverse Laplacian which is well defined, indeed we can associate a bi-linear form to the Laplacian and show through the Poincaré inequality the fact that $\Delta$ is one to one. We can formally define $\Delta^{-1}$ using Green functions.

**Remark 1** : On the 2-flat Torus $\mathbb{T}^2$, the Fourier transform is very useful as all functions (velocity components, pressure, ...) can be expressed as Fourier series (trigonometric polynomials). The derivatives then become simple algebraic operations in Fourier space, i.e. a multiplication with the wave number. This allows an easy and inexpensive inversion of the Laplacian by a simple division in Fourier space in order to solve the Poisson equation for the stream function. Moreover, thanks to the FFT (Fast Fourier Transform) the computational complexity is almost optimal ($O(N \log N)$).

## 2.1 Properties of the Navier-Stokes and Euler equations

The Euler equations are a system of conservation laws which means that the mass and the momentum of the fluid flow are exactly preserved. Moreover, we also have for $\boldsymbol{u}_0, \omega_0 \in L^2(\mathbb{T}^2)$ a stability, i.e. $\boldsymbol{u}(\cdot, t), \omega(\cdot, t) \in L^2(\mathbb{T}^2)$ for all $t \geq 0$. The $L^2$ norm is related to physical quantities, like energy and enstrophy which satisfy balance equations. In order to verify the numerical implementation of the Characteristic Mapping method, there are several ways of computing different quantities such as the energy, the enstrophy and also the palinstrophy, which give us important information on the behaviour and the reliability of the numerical simulations. This part defines these quantities and their balance equations, as well as ways to compute them such that formally or numerically, we obtain the same results.

### 2.1.1 Balance equations

In order to define the energy, enstrophy and palinstrophy, we need to consider respectively the $L^2$ norms of the velocity, vorticity and the vorticity gradients. Formally we have:

$$E(t) = \frac{1}{2}\int_{\mathbb{T}^2} |\boldsymbol{u}(\boldsymbol{x}, t)|^2 d\boldsymbol{x} \tag{8}$$

$$Z(t) = \frac{1}{2}\int_{\mathbb{T}^2} |\omega(\boldsymbol{x}, t)|^2 d\boldsymbol{x} \tag{9}$$

$$P(t) = \frac{1}{2}\int_{\mathbb{T}^2} |\nabla\omega(\boldsymbol{x},t)|^2 d\boldsymbol{x} \tag{10}$$

The conservation of equation (8) and (9) will be a main aspect of study. Equation (10) will also be useful since for high Reynolds number or inviscid fluids flow, the vorticity develops strong gradients which can be quantified by the palinstrophy.

The energy and enstrophy dissipation in the case of Navier-Stokes are given by the following formulas [Boffetta and Ecke, 2012] :

$$\begin{cases} \dfrac{d}{dt}E(t) = -2\nu Z(t) \\ \dfrac{d}{dt}Z(t) = -2\nu P(t) \end{cases} \tag{11}$$

These balance equations can be deduced from (1) and (4) by multiplying the momentum equation by $\boldsymbol{u}$ and $\omega$, respectively. The $L^2$ norm of $\boldsymbol{u}, \omega$ can then be obtained by integration by parts. Since the Euler equations are valid in the case of inviscid fluid, i.e. with the kinematic viscosity $\nu$ being zero, we deduce that :

$$\frac{d}{dt}E(t) = \frac{d}{dt}Z(t) = 0$$

Hence in the inviscid case the energy dissipation $dE/dt$ and the enstrophy dissipation $dZ/dt$ are zero, this implies that the energy and the enstrophy are both conserved in two dimensional flows. Solving the Euler equations numerically, we expect that these quantities are not conserved exactly anymore, which is part of numerical tests we are going to perform later in the report.

### 2.1.2 Isotropic energy spectrum

For any time $t \geq 0$ the velocity $\boldsymbol{u}(\cdot,t)$ and the vorticity $\omega(\cdot,t)$ are periodic on $L^2(\mathbb{T}^2)$. Hence both can be transformed into a Fourier series and Parseval's theorem can be applied, which yields another expression for computing energy and enstrophy.

Before proceeding further, we need to define the 2D Fourier transform $\mathcal{F}$ and its inverse $\mathcal{F}^{-1}$. Since $\mathbb{T}^2$ is bounded, we have $L^1(\mathbb{T}^2) \cap L^2(\mathbb{T}^2) = L^2(\mathbb{T}^2)$, from here we can define :

$\forall f \in L^2(\mathbb{T}^2)$ and $\boldsymbol{k} \in \mathbb{Z}^2$, we define :

$$\mathcal{F}(f)(k) = \widehat{f}(\boldsymbol{k}) = \int_{\mathbb{T}^2} f(\boldsymbol{x}) \exp(-2i\pi\boldsymbol{k}\cdot\boldsymbol{x}) d\boldsymbol{x} \tag{12}$$

The inverse transform is given by:

$$\mathcal{F}^{-1}(\widehat{f})(x) = f(x) = \sum_{\boldsymbol{k}\in\mathbb{Z}^2}\sum \widehat{f}(\boldsymbol{k}) \exp(2i\pi\boldsymbol{k}\cdot\boldsymbol{x}) \tag{13}$$

For a flow which is homogeneous in space, a spectral description is appropriate and allows us to examine the flow properties as a function of wave number. Equation (8), (9) and (10) are

made over real space, but using the Parseval identity allows the computation to be made in Fourier space :

$$E(t) = \frac{1}{2} \sum_{k_x \in \mathbb{Z}} \sum_{k_y \in \mathbb{Z}} |\widehat{\boldsymbol{u}}(\boldsymbol{k}, t)|^2$$
$$Z(t) = \frac{1}{2} \sum_{k_x \in \mathbb{Z}} \sum_{k_y \in \mathbb{Z}} |\widehat{\omega}(\boldsymbol{k}, t)|^2$$
$$P(t) = \frac{1}{2} \sum_{k_x \in \mathbb{Z}} \sum_{k_y \in \mathbb{Z}} |\widehat{\nabla\omega}(\boldsymbol{k}, t)|^2$$

For $\boldsymbol{x} = (x, y) \in \mathbb{T}^2$ and $\boldsymbol{k} = (k_x, k_y) \in \mathbb{Z}^2$. We express $\widehat{\boldsymbol{u}}$ as follows :

$$\widehat{\boldsymbol{u}}(\boldsymbol{k}, t) = \widehat{\boldsymbol{u}}(k_x, k_y, t) = \int_0^{2\pi} \int_0^{2\pi} \boldsymbol{u}(x, y, t) \exp(-2i\pi(xk_x + yk_y))dxdy$$

One way to quantify the evolution of spatial scales is through the Fourier expansion of the solution and the decay of the Fourier coefficients with wave number. We can define the isotropic energy spectrum for a fixed time $t > 0$ :

$$E(k) = \sum_{k-1/2 \leq |\boldsymbol{k}| < k+1/2} \sum |\widehat{\boldsymbol{u}}(\boldsymbol{k})|^2$$

with $|\boldsymbol{k}| = k \in \mathbb{N}$.

The enstrophy spectrum is defined correspondingly via the vorticity, $\omega$:

$$Z(k) = \sum_{k-1/2 \leq |\boldsymbol{k}| < k+1/2} \sum |\widehat{\omega}(\boldsymbol{k})|^2$$

Using the incompressiblity property, we can also express the enstrophy spectrum in terms of the energy spectrum, $E(k)_{k \in \mathbb{Z}}$ :

$$Z(k) = k^2 E(k)$$

The method above computes the enstrophy spectrum and the energy spectrum by integrating over concentric circles of radius $|\boldsymbol{k}| = \sqrt{k_x^2 + k_y^2} = k$ in the spectral space. This spectrum will be helpful in order to look into the decay of the magnitude of high wave number coefficients of the vorticity and is motivated by the fact that turbulence is statistically isotropic

Similarly, we can do the same for the palinstrophy, but its study is not as revelant as the two spectra mentioned above.

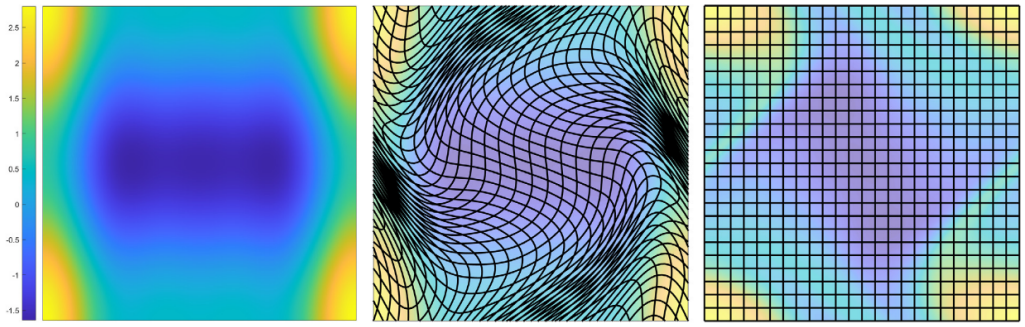
**Remark 2** : 2D-turbulence for low viscosity is characterized by the conservation of energy, the finite dissipation of enstrophy and the exponential increase in palinstrophy [Lesieur, 1987]. For vanishing viscosity, we have conservation of energy and enstrophy, but exponential growth of palinstrophy, which makes the computation of inviscid flows very challenging, as vorticity gradients grow over time.

## 2.2 Flow map

The flow map, also called the characteristic map, is the backbone of the Characteristic Mapping Method which evolves the deformation map of the domain generated by the velocity field associated to the fluid. This approach is geometric and exploits the structure of the fluid flow by constructing a numerical characteristic map. It traces a point on a given characteristic curve back to its initial position through a spatial transformation. This "pullback" is made possible thanks to a suitable assumption on the smoothness of the velocity $\boldsymbol{u}$ being a diffeomorphism in the case of the incompressible Euler equations.

Given a diffeomorphism $X : \mathbb{T}^2 \times \mathbb{R}_+ \to \mathbb{T}^2$ in the first variable, i.e. a bicontinuous and continously differentiable application such that for any initial point $X(\boldsymbol{x}, t = 0) = \boldsymbol{x}$ under the velocity field $\boldsymbol{u}$, we have for any characteristic curve :

$$X(\boldsymbol{x}(t), t) = \boldsymbol{x}(0)$$



(a) Initial condition. (b) Domain deformation from characteristic map. (c) Pullback of the initial condition by the map.

Figure 1: Pullback of the map at time $t$ to its initial condition[1].

The smoothness of the characteristic curve and the property of reversibility thanks to the existence of an inverse for the diffeomorphism allows the pullback.

Figure 1 shows the evolution of the initial condition through the deformation of the domain (a), (b) and the result of the pullback to the initial condition(c).

We can decompose the flow map into several pieces, thus $\forall n \in \mathbb{N}$, $T_f > 0$ and $(\tau_i)_{0 \leq i \leq T_f}$ $\tau_1 < \tau_2 < ... < \tau_n$ with $\sum_{i=1}^{n} \tau_i = T_f$ :

$$X_0(x, T_f) = X_1(X_2(...X_n(x, \tau_n)...), \tau_2), \tau_1)$$

with each sub map $(X_i)_{1 \leq i \leq n}$ solving the following problem :

$$\partial_t X_i + \boldsymbol{u} \cdot \nabla X_i = 0 \quad \forall t \geq \tau_{i-1}, x \in \mathbb{T}^2 \tag{15}$$

$$X_i(x, \tau_{i-1}) = x \tag{16}$$

For any characteristic curve $\gamma$, we define the following diffeomorphism called the backward caracteristic maps :

$$\begin{aligned} \boldsymbol{X} : \mathbb{T}^2 \times \mathbb{R}_+ &\to \mathbb{T}^2 \\ (\gamma(t), t) &\to \boldsymbol{X}(\gamma(t), t) = \gamma_0 \end{aligned}$$

[1]Image taken from Yin et al. [2021a]

The smoothness of the characteristic curve in the case of incompressible Euler flows allows $\boldsymbol{X}$ to be a diffeomorphism, that is crucial to trace back the particles to their initial positions for example.

As we will explain on later, the CM method being a semi-Lagrangian formulation, we need to translate our equation from the Eulerian to the Lagrangian point of view and this change will be more visible as we involve the backward map.

# 3 The Characteristic Mapping Method

The CM method is a semi-Lagragian approach which has been developed for solving the 2D Euler equations by Yin et al. [2021a] and then extended to the 3D case by Yin et al. [2021b]. In the 2D Euler case, it consists of solving the vorticity transport equation. The vorticity is evolved under the velocity field $\boldsymbol{u}$, which is determined through the Biot-Savart law from the vorticity.

## 3.1 Formal aspect

### 3.1.1 Framework

Let's consider an advection equation with a divergence free velocity field $\boldsymbol{u}$ :

$$\begin{cases} \partial_t \omega + (\boldsymbol{u} \cdot \nabla)\omega & = 0 \\ \omega(\boldsymbol{x}, 0) & = \omega_0(\boldsymbol{x}) \end{cases} \tag{17}$$

Advection equations are hyperbolic, as such, the solution at each points travels along characteristic curves $\gamma : \mathbb{R}_+ \to \mathbb{T}^2$, on which $\omega$ is constant :

$$\frac{d}{dt}\omega(\gamma(t), t) = 0 \Leftrightarrow \omega(\gamma(t), t) = \omega_0(\gamma(t))$$

The characteristic curves and the velocity field are given by the following Cauchy problem :

$$\begin{aligned} \frac{d}{dt}\gamma(t) &= \boldsymbol{u}(x, t) \\ \gamma(0) &= \gamma_0 \end{aligned}$$

The global existence and uniqueness of a characteristic curve initial problem assures that the map is bijective. In the case of the incompressible Euler equations, the characteristic curves are smooth [Wolibner, 1933], thus the backward map $\boldsymbol{X}$ is a diffeomorphism and can be computed by solving the following equation :

$$\begin{cases} \partial_t \boldsymbol{X} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{X} & = 0 \\ \boldsymbol{X}(\boldsymbol{x}, 0) = \boldsymbol{x} \end{cases} \tag{18}$$

We have seen above that the characteristic map $\boldsymbol{X}(\cdot, t)$ takes a point $\boldsymbol{x}(t)$ at time $t$ and returns it to the initial position $\boldsymbol{x}_0$ of this curve, allowing us to write the solution to the advection problem as follows:

$$\omega(\boldsymbol{x}, t) = \omega_0(\boldsymbol{X}(\boldsymbol{x}, t)) \tag{19}$$

Equation (19) is valid for any initial condition $\omega_0$ and the solution remains $\omega_0 \circ \boldsymbol{X}$ as the backward map is independent of the transported quantity and only captures the deformation made by the velocity field $\boldsymbol{u}$. This is an important step of the CM method because from here on out, we totally embrace the Lagrangian point of view which is symbolized by looking at the map deformation.

Henceforth, we can combine the previous steps. More precisely equation (7), being the velocity expressed through the Biot-Savart law, equation (18) representing the flow map $\boldsymbol{X}$ and finally equation (19) which expresses in our case the vorticity $\omega$ as function of the deformation map $\boldsymbol{X}$. Altogether, these equations form a system called advection-vorticity coupling :

$$\begin{cases} \omega(\boldsymbol{x}, t) & = \omega_0(\boldsymbol{X}(x, t)) \\ \boldsymbol{u} & = \Delta^{-1}\nabla^{\perp}\omega \\ (\partial_t + \boldsymbol{u} \cdot \nabla)X & = 0 \end{cases} \tag{20}$$

The incompressibility condition imposes a divergence free velocity $\nabla \cdot \boldsymbol{u} = 0$, it can also be expressed into a Lagrangian point of view equivalent to the incompressibility constraint which also involves the Jacobian of the deformation map $\boldsymbol{X}$ :

$$det(\nabla \boldsymbol{X}) = 1 \tag{21}$$

which implies that the Jacobian of the flow map needs to be an isometry imposing a volume preserving property. This is the Lagrangian equivalent of the Eulerian counterpart $\nabla \cdot \boldsymbol{u} = 0$.

### 3.1.2 Semi-group structure

The interesting feature of the flow map is its semi-group structure, we recall the basic property of a semi-group below for a formal Banach space :

**Definition 1** (Semi-group operators)**.** *Let $B$ be a Banach space. A semi-group on $B$ is a family $(T_t)_{t\geq 0}$ of bounded linear operator such that :*

$T_0 = Id$

$T_s \circ T_t = T_{s+t} \quad$ *for all* $s, t \in \mathbb{R}_+$

The semi-group structure of the flow map is important for the CM method, instead of evolving the characteristic map through the system of ODEs, we define the evolution of the characteristic map through the composition of the map. This allows us to break the map into several pieces we call sub-map, hence allowing us to store each separately and solving each of them.

**Definition 2** (Sub-map)**.** *We call characteristic sub-map of the time interval $[t_0, t_f]$ with $0 \leq t_0 \leq t_f$ the characteristic transformation of the domain $\mathbb{T}^2$ mapping $t_f$ to $t_0$, formally we define :*

$$\boldsymbol{X}_{[t_f, t_0]} : \mathbb{T}^2 \to \mathbb{T}^2$$

*such that :*

$$\partial_t \boldsymbol{X} + \boldsymbol{u} \cdot \nabla \boldsymbol{X} = 0 \tag{22a}$$

$$\boldsymbol{X}(\boldsymbol{x}, t_0) = \boldsymbol{x} \tag{22b}$$

$$\boldsymbol{X}_{[t_f, t_0]}(\boldsymbol{x}) := \boldsymbol{X}(\boldsymbol{x}, t_f) \tag{22c}$$

## 3.2 Numerical scheme and implementation

In this section we present the numerical scheme and focus on the key features and the implementation, which is illustrated in the flowchart 7.2. Further details on the method and the CUDA code, which has been developed during the master thesis of Yadav [Yadav, 2015]. In Yin et al. [2021a] the method is analyzed in detail numerically and validation tests are done using a MATLAB implementation which differs from the CUDA code. From now on, We distinguish the theoretical flow map $\boldsymbol{X}$ from the numerical characteristic map $\chi$.

The flow map $\chi$ is evolved by the gradient augmented level set method (GALS) which advects particles starting on a Cartesian grid backwards in time using stencil of $\epsilon$-points and tracing back their respective foot points (see e.g. Kolomenskiy et al. [2016]). The numerical implementation is detailed in algorithm (1) below. After the map advection, we proceed with a Hermite interpolation of the vorticity and the computation of the velocity field with the Biot-Savart kernel in Fourier space, before starting with the remapping process.

### 3.2.1 Remapping

The remapping process aims to adaptively resolve the increasingly complicated spatial features that ought to appear due to the absence of a regularizing viscosity term. Since the Euler equations develop small scale spatial features, the characteristic map computed on a fixed coarse grid is only valid for a limited period of time. Afterwards the resolution requirements are not sufficient anymore. We measure the quality of the characteristic map by looking at the incompressibility condition, which can be expressed as the determinant of the Jacobian of the characteristic map. Hence by introducing an arbitrary but reasonable threshold $\delta$, we define an incompressibility check :

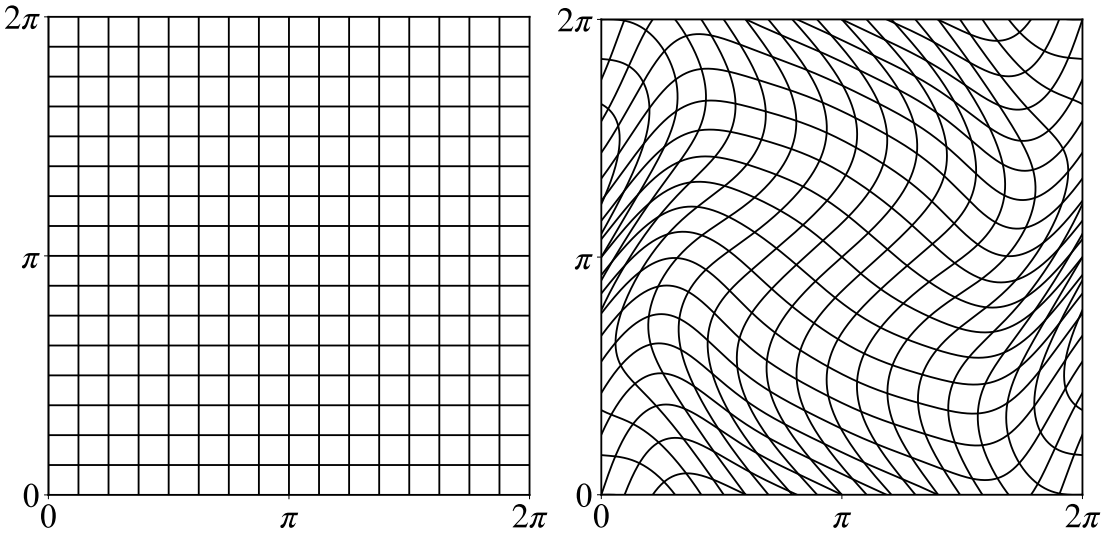
$$||det(\nabla\chi) - 1||_\infty \leq \delta \tag{23}$$

When equation (23) does not hold anymore, the remapping process starts as the map quality is judged not to be accurate enough. This step can also be seen in the flowchart 7.2.

The semi-group property of the characteristic maps implies that maps can be decomposed to form another map, i.e. for two numerical characteristic maps $\chi_1, \chi_2 : \mathbb{T}^2 \times \mathbb{R}_+ \to \mathbb{T}^2$ :

$$\chi = \chi_2 \circ \chi_1$$

The composition of two cubic maps yields a sixth order polynomial in each variable. We can form another cubic map for the composition by taking a projection which can then be computed on a much finer grid allowing us to retain more sub-map features. This also means that each time the global approximation error grows beyond a certain point, we can resample the map on a finer grid and represent the solution on a finer grid than the one used for the computation of $\chi$.




(a) Initial map with identity (b) Deformation of the map at time $t = 1$

Figure 2: Flow map at two different time steps.

Figure 2 shows an example of a flow map at two different time steps. We start with a Cartesian grid which represents the identity transformation and then we can observe the deformation of the grid by the flow. Each deformation is a reversible process which is an important property for the remapping process.

### 3.2.2 Grids and Multiscale Representation

Different length scales are present and develop in the solution during the flow evolution and the simulation is limited by the available computational resources. Thus we need to make choices and choose an appropriate degree of spatial truncation for each evolved quantity : The flow-map $\chi$, the stream function $\psi$, the velocity $\boldsymbol{u}$ and the vorticity $\omega$ are represented on two grids with different resolution depending on their importance in the global evolution of the flow. Using the fact that $\boldsymbol{u}$ is smoother than the vorticity field due to the Biot-Savart law, we can save some resources and compute it on a uniform coarse grid $X_g \in \mathbb{M}_n(\mathbb{R})$ where $n^2 \in \mathbb{N}$ is the resolution of the grid and $2\pi/n$ its spatial step. Furthermore, assuming that the large scales of the velocity drive the dynamics of the flow, it is important to save the small scale features of the vorticity by transporting it on a grid $X_f \in \mathbb{M}_m(\mathbb{R})$ where $m \in \mathbb{N}$, and $m > n$ which allows for finer spatial steps and finer scales to be saved.

In the CUDA code, grids are represented in an "Hermitian" array, i.e. grids are saved on an array along with their derivatives $\partial_x, \partial_y$ and cross derivatives $\partial^2_{xy}$.

### 3.2.3 Lagrange interpolation of the velocity

For time integration using Runge-Kutta schemes, the velocity field is required at intermediate time steps. To this end we implemented a 3rd order Lagrange time interpolation in the CUDA code. This allows to evaluate the velocity field at any intermediate time instant. Thus Runge-Kutta schemes can be used. Let's define Lagrange coefficients on the stencil $\{t_n, t_{n-1}, t_{n-2}\}$ in function of $t > 0$ :

$$l_1, l_2, l_3 : \mathbb{R}_+ \to \mathbb{R}$$

with:

$$l_0(t) = \frac{(t - t_{n-1})(t - t_{n-2})}{(t_n - t_{n-1})(t_n - t_{n-2})}$$
$$l_1(t) = \frac{(t - t_n)(t - t_{n-2})}{(t_{n-1} - t_n)(t_{n-1} - t_{n-2})}$$
$$l_2(t) = \frac{(t - t_n)(t - t_{n-1})}{(t_{n-2} - t_n)(t_{n-2} - t_{n-1})}$$

Hence, we can define $\tilde{\boldsymbol{u}}$ on the grid $X_g$:

$$\tilde{\boldsymbol{u}} : \mathbb{R}^2 \times \mathbb{R}_+ \to \mathbb{R}^2 \tag{24}$$

Thus, for $\boldsymbol{x} = (x, y) \in \mathbb{T}^2$ and $t_i$ which extends the previous velocity on the grid $X_g$ we have:

$$\tilde{\boldsymbol{u}}_{|X_g}(\cdot, t_i) = \boldsymbol{u}_{|X_g}(\cdot, t_i)$$

and this being defined as following :

$$\tilde{u}(\boldsymbol{x}, t) = \sum_{i=0}^{2} l_i(t)\boldsymbol{u}(\boldsymbol{x}, t_{n-i}) \tag{25}$$

One can note that for each fixed time $t$, $\tilde{u}$ is a linear combination of $\boldsymbol{u}_{n-i}$ for $0 \leq i \leq 2$. It hence conserves the divergence free property of $\boldsymbol{u}$ i.e.:

$$\nabla \cdot u_{n-i} = 0 \qquad 0 \leq i \leq 2 \implies \nabla \cdot \tilde{\boldsymbol{u}} = 0$$

### 3.2.4 Time discretization

The current implementation of the code contains three different time schemes for the backwards time integration:

- explicit backward Euler
- Adams-Bashforth of order 2 (backwards) + fixed point iteration(Adams-Moulton)
- Runge-Kutta 3rd order (backwards) (Main one)

Once the current velocity $\boldsymbol{u}$ is computed on the coarse grid $\boldsymbol{X}_g$ and previous velocities are saved for interpolation, we can use a third order backwards Runge-Kutta scheme to advect the numerical map $\chi$ :

**Algorithm 1** Map advection $\chi(\cdot, t_n) \to \chi(\cdot, t_{n+1})$

**Data:**
$\epsilon = 10^{-6}$
$\{\epsilon_x^0, \epsilon_x^1, \epsilon_x^2, \epsilon_x^3\} = \{\epsilon, \epsilon, -\epsilon, -\epsilon\}$
$\{\epsilon_y^0, \epsilon_y^1, \epsilon_y^2, \epsilon_y^3\} = \{\epsilon, -\epsilon, \epsilon, -\epsilon\}$
**for each** $\boldsymbol{x} = (x, y) \in X_g$ **do**

**for each** $k \in [0, 1, 2, 3]$ **do**
define $\boldsymbol{x}_\epsilon^k = (x + \epsilon_x^k, y + \epsilon_y^k)$

Compute foot point : $\boldsymbol{x}_f^k$

**end for**

Update :

$$\left.\begin{aligned}
\chi(\boldsymbol{x}, t_{n+1}) &= \frac{1}{4}\left(\chi(x_f^0, t_n) + \chi(x_f^1, t_n) + \chi(x_f^2, t_n) + \chi(x_f^3, t_n)\right) \\
\partial_x\chi(\boldsymbol{x}, t_{n+1}) &= \frac{1}{4\epsilon}\left(\chi(x_f^0, t_n) + \chi(x_f^1, t_n) - \chi(x_f^2, t_n) - \chi(x_f^3, t_n)\right) \\
\partial_y\chi(\boldsymbol{x}, t_{n+1}) &= \frac{1}{4\epsilon}\left(\chi(x_f^0, t_n) - \chi(x_f^1, t_n) + \chi(x_f^2, t_n) - \chi(x_f^3, t_n)\right) \\
\partial_x\partial_y\chi(\boldsymbol{x}, t_{n+1}) &= \frac{1}{4\epsilon^2}\left(\chi(x_f^0, t_n) - \chi(x_f^1, t_n) - \chi(x_f^2, t_n) + \chi(x_f^3, t_n)\right)
\end{aligned}\right\} \tag{26}$$

**end for**

Instead of computing foot points for each grid point $\boldsymbol{x} = (x, y) \in X_g$, we define a set of $\epsilon-$points $\boldsymbol{x} + (\epsilon, \epsilon), \boldsymbol{x} + (-\epsilon, \epsilon), \boldsymbol{x} + (\epsilon, -\epsilon), \boldsymbol{x} + (-\epsilon, -\epsilon)$ which form an $\epsilon$-cell around $\boldsymbol{x}$

The characteristic map can be evolved using its semi-group property. For the current time $t_n = n\Delta t$, we have :

$$\boldsymbol{X}_{[t_{n+1}, t_0]} = \boldsymbol{X}_{[t_n, t_0]} \circ \boldsymbol{X}_{[t_{n+1}, t_n]}$$

We extend the velocity $\boldsymbol{u}$ by using an extrapolation with a third order Lagrange polynomial in time, described in section (3.2.3).

By defining the four epsilon points $x_\epsilon^k$ with $0 \leq k < 4$ as an $\epsilon$-cell we represent our stencil. Characteristics that go through $x_\epsilon^0, x_\epsilon^1, x_\epsilon^2, x_\epsilon^3$ are traced backwards in time from $t_{n+1}$ to $t_n$.

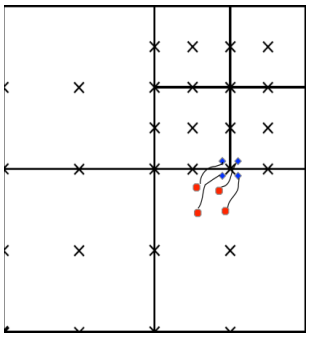


Figure 3: Grid point with its $\epsilon$-cell (blue points) and its respective foot points (red points)[2]

We use an third order explicit Runge-Kutta scheme of the velocity $\boldsymbol{u}$ backwards in time to approximate the sub map $\boldsymbol{X}_{[t_{n+1},t_n]}$:

$$X_{[t_{n+1},t_n]}(\boldsymbol{x}) = \boldsymbol{x} - \Delta t \sum_{j=1}^{3} b_j k_j \tag{27}$$

with $(k_j)_{1\leq j\leq 3}$ being defined as follows :

$$k_j = \boldsymbol{u}(\boldsymbol{x} - \Delta t \sum_{m=1}^{j-1} a_{jm} k_m, t_{n+1} - c_j \Delta t) \qquad \forall 1 \leq j \leq 3 \tag{28}$$

where the coefficients $(a_j), (b_j)$ and $c_j$ can be found in the following Butcher table :

$$\begin{array}{c|ccc} c_1 & a_{11} & a_{12} & a_{13} \\ c_2 & a_{21} & a_{22} & a_{23} \\ c_3 & a_{31} & a_{32} & a_{33} \\ \hline & b_1 & b_2 & b_3 \end{array} = \begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 1 & -1 & 2 & 0 \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array}$$

Hence, we have for $(k_j)$ and a negative time step $-\Delta t$ :

- $k_1 = (k_{1x}, k_{1y}) = \tilde{\boldsymbol{u}}(\boldsymbol{x}, t_{n+1})$
- $k_2 = (k_{2x}, k_{2y}) = \tilde{\boldsymbol{u}}\left(\boldsymbol{x} - k_1 \frac{\Delta t}{2}, t_{n+1} - \frac{\Delta t}{2}\right)$
- $k_3 = (k_{3x}, k_{3y}) = \tilde{\boldsymbol{u}}\left(\boldsymbol{x} + k_1 \Delta t - 2k_2 \Delta t, t_n\right)$

Thus, we trace back the foot point :

$$x_f = x_\epsilon - \Delta t \frac{k_{1x} + 4k_{2x} + k_{3x}}{6}$$
$$y_f = y_\epsilon - \Delta t \frac{k_{1y} + 4k_{2y} + k_{3y}}{6}$$

The map advection is done through equation (26) and using an averaging for the first equation and a second order centered scheme for computing the derivatives. Given a suitable assumption on the smoothness of the characteristic map, $\boldsymbol{X}$, we can derive the system (26), i.e. for

[2]Image taken from [Kolomenskiy et al., 2016].

$\boldsymbol{X} \in \mathcal{C}^2(\mathbb{T}^2 \times \mathbb{R}_+, \mathbb{T}^2)$, it is possible to expand the values of $\boldsymbol{X}$ at the corner of the $\epsilon$-cell into a Taylor series and to obtain a second order method from the finite size of $\Delta X$.

The error can be further reduced by using a stencil larger than $\{\boldsymbol{x}_f^0, \boldsymbol{x}_f^1, \boldsymbol{x}_f^2, \boldsymbol{x}_f^3\}$ in order to achieve a third/fourth order discretization and to increase the value of $\epsilon$ from $10^{-6}$ to $10^{-4}$. This increase in the order of $\epsilon$ should improve the number of remappings and also allows us to be further away from the machine precision which is $2.22044605 \times 10^{-16}$ for both the RTX 3060 and for the TESLA V100S GPU. The machine precision was obtained by running the standard machine epsilon code on one CUDA thread :

**Algorithm 2** Machine epsilon

**Data:**
$\epsilon = 1$
**while** $(1.0 + 0.5 * \epsilon)$ $\neq 1.0$ **do**
  $\epsilon = 0.5 * \epsilon$
**end**

#### 3.2.5 Computation of derivatives through FFT

The Fast Fourier Transform (FFT) is used in the code in order to compute several quantities such as $\omega$ and its derivatives or to invert the Laplacian and to compute $\psi$ and subsequently $\boldsymbol{u}$. The FFT has the ability to be parallelized, thus being faster while also coming with the advantage to not introduce any numerical error compared to a finite difference calculation.

By defining $\omega(\boldsymbol{x}, t_n) = \omega_n(\boldsymbol{x})$ and $\psi(\boldsymbol{x}, t_n) = \psi_n(\boldsymbol{x})$, we use the following formula :

$$
\begin{aligned}
\psi_n(\boldsymbol{k}) &= -\widehat{\omega_n}(\boldsymbol{k})/k^2 & \psi_n(\boldsymbol{x}) &= \mathcal{F}^{-1}(\widehat{\psi}_n(\boldsymbol{k})) \\
\partial_x \widehat{\psi}_n(\boldsymbol{k}) &= ik_x \widehat{\psi}_n(\boldsymbol{k}) & \partial_x \psi_n(\boldsymbol{k}) &= \mathcal{F}^{-1}(\partial_x \widehat{\psi}_n(\boldsymbol{k})) \\
\partial_y \widehat{\psi}_n(\boldsymbol{k}) &= ik_y \widehat{\psi}_n(\boldsymbol{k}) & \partial_y \psi_n(\boldsymbol{k}) &= \mathcal{F}^{-1}(\partial_y \widehat{\psi}_n(\boldsymbol{k})) \\
\partial^2_{xy} \widehat{\psi}_n(\boldsymbol{k}) &= -k_x k_y \widehat{\psi}_n(\boldsymbol{k}) & \partial^2_{xy} \psi_n(\boldsymbol{k}) &= \mathcal{F}^{-1}(\partial^2_{xy} \widehat{\psi}_n(\boldsymbol{k}))
\end{aligned}
$$

In the CUDA code, we compute the velocity from the vorticity data using a parallelized IFFT based on these formula :

## 4 Fluid particles and inertial particles

In this section we will describe the implementation of the fluid and inertial point particles using for the later the Maxey-Riley model. The particles are advected by the fluid flow and different values of inertia can be considered.

### 4.1 Fluid Particles

The goal is to compute the transport of particles driven by the fluid velocity. The fluid particle is a particle without inertia and satisfies an equation of motion from which we need the fluid velocity $\boldsymbol{u}$ at the particle position. Thus it follows directly the fluid flow.

Given a grid, a particle position $\boldsymbol{x} = (x, y) \in [0, 2\pi]^2$ and $t \geq 0$, we need to find an approximation of the velocity $\boldsymbol{u}(\boldsymbol{x}, t)$ which is only known on the grid points. The first step consists of

finding the four points of the grid such that $\boldsymbol{x}$ can be written as a convex combination of these four points. In others words, given a grid $X_g$ represented by a square matrix in $\mathcal{M}_n(\mathbb{R})$ with $n^2 \in \mathbb{N}$ being the resolution by partitioning $[0, 2\pi]^2$ into $n^2$ blocks, we need to find the blocks that contain the particle at coordinate $\boldsymbol{x} = (x, y)$.

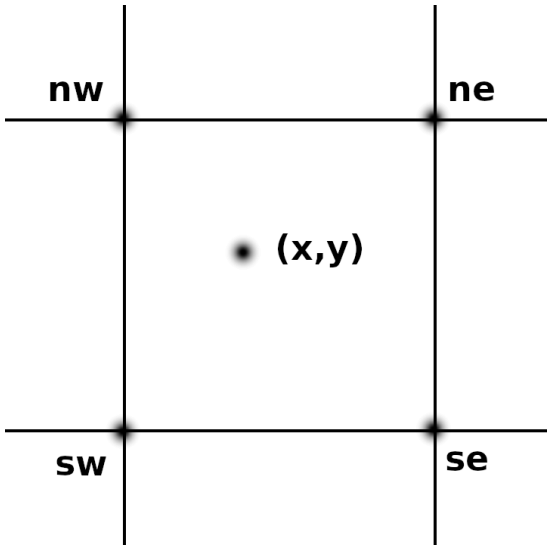



Figure 4: Cardinal points on coarse grid.

Let's denote by se, sw, ne, nw the four cardinal points, as illustrated in figure 4, of the grid around $(x, y)$. Then by using a cubic Hermite interpolation on the stencil (se, sw, ne, nw) given a fixed time $t \in \mathbb{R}_+$, we can approximate $\psi(\boldsymbol{x}, t)$ and as such we can also deduce the velocity field $\boldsymbol{u} = (u_1, u_2) = (\partial_y \psi, -\partial_x \psi)$.

The differential equation to solve for the particle transport is the following :

$$\frac{d}{dt}\boldsymbol{x} = \boldsymbol{u} \tag{29}$$

By using an explicit Euler scheme and by introducing $t_n = n\Delta t$ and $\boldsymbol{u}_n(\boldsymbol{x}) := u(\boldsymbol{x}, t_n) = (\psi_y(\boldsymbol{x}, t_n), -\psi_x(\boldsymbol{x}, t_n))$ with $\boldsymbol{x} = (x, y) \in \mathbb{T}^2$, we can define the following problem :

$$\begin{cases} x_{n+1} = x_n + \Delta t \partial_y \psi \\ y_{n+1} = y_n - \Delta t \partial_x \psi \end{cases} \tag{30}$$

It is possible to reduce the error in the numerical solution further by using instead of explicit Euler a second order Runge-Kutta scheme :

$$k_1^n = \Delta t \psi(u_n, t_n), \qquad k_2^n = \Delta t \psi\left(u_n + \frac{k_1^n}{2}, t_n + \frac{\Delta t}{2}\right)$$

Hence the particle advection can be written as :

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \frac{1}{2}(k_1^n + k_2^n) \tag{31}$$

The initial particle position $\boldsymbol{x}_0$ is following a uniform distribution on the square $[0, 2\pi]^2$.

We put fluid particles into a mixed layer initial condition which is explained further down in section 5.1. The simulation computed 51,200 time step $\Delta t = 1/1024$, on a coarse grid of resolution $128^2$, a fine grid of resolution $512^2$, an incompressibility threshold $\delta = 10^{-4}$ and $N_p = 131072$ particles. Furthermore we provide through figure 5 and 6 a visual representation of the underlying vorticity and the advection of 6554 particles at time $t = 1, t = 30$ and $t = 50$ respectively.

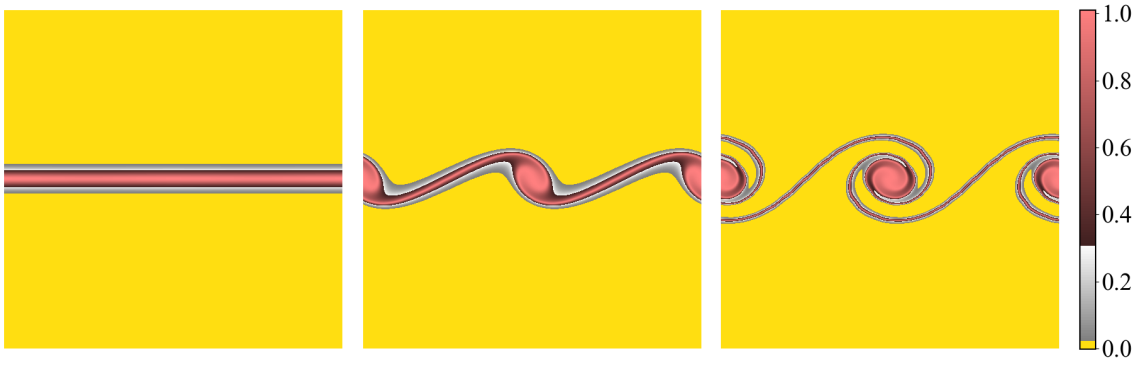
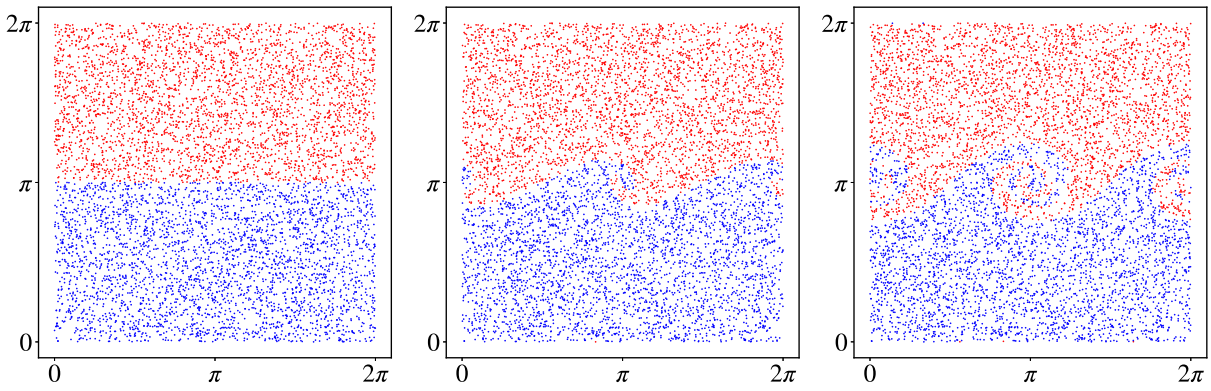



Figure 5: Vorticity for the single shear layer at time $t = 1, 35$ and 50



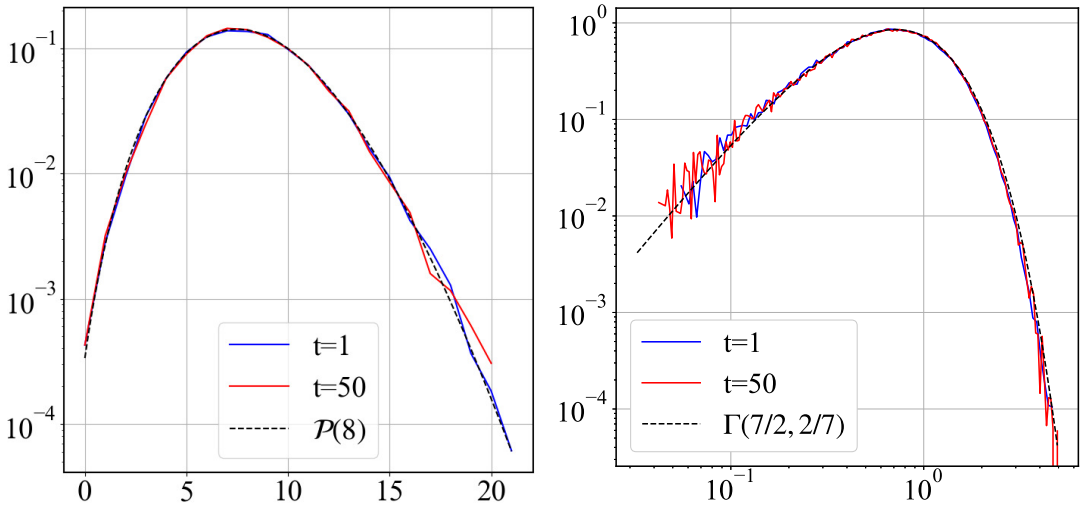

(a) $t = 1$ (b) $t = 35$ (c) $t = 50$

Figure 6: Fluid particle advection on a $128^2$ coarse grid at time $t = 1$, 35 and $t = 50$.

In figure 6, the particles are either blue or red depending on their initial position of the vertical component, i.e. blue particles $\boldsymbol{x}_b = (x_b, y_b)$ are such that $y_b < \pi$ and similarly red particles $\boldsymbol{x}_r$ such that $y_r > \pi$. Using two colors we can better visualize the mixing of the two layers around the high vorticity region which are at the interface between the layers.

Since the particles are initially uniformly random distributed on $[0, 2\pi]^2$ and due to the incompressibility condition, we should expect from the law of large numbers a density which follows a Poisson law with parameters $N_p/n^2$ where $N_p$ represents the number of particles in the simulation, $n^2$ represents the resolution of the grid where the particles are advected. The density should follow a Poisson distribution throughout the simulation. This can be indeed observed in Figure 7.

Figure 7 shows at time $t = 1$ and $t_{final} = 50$ the PDF the particles density (left) and the PDF the volume of Voronoi cells (right). We added in dashed lines the theoretical PDFs for both distributions, i.e. the Poisson distribution $\mathcal{P}(\lambda)$ of parameter $\lambda = 8$ for the density and the gamma distribution $\Gamma(k, \theta)$ of parameters $k = 7/2$ and $\theta = 2/7$ [Ferenc and Néda, 2007] for Voronoi volumes which represents a partition of the plane allowing us to attribute a volume




(a) Density (b) Voronoi

Figure 7: PDF of the density and the Voronoi volume computed for fluid particle at time $t = 1$ and $t = 50$. The dashed lines represent the Poisson (left) and the Gamma (right) distribution.

to each particles. The density has been computed with bins of size $(2\pi/128)^2$ which was the size of the resolution of the simulation. In the case of volume of Voronoi cells, there is no need for the resolution parameter. We can observe for both PDFs, a good agreement with the theoretical distributions which allows us to say that the particles stay randomly distributed and the flow is thus volume preserving.

### 4.2 Inertial Particle Advection

Now we consider point particles with inertia, which do not necessarily follow the fluid flow anymore. In order to simulate the drag force of an inertial particle within the fluid, we need to append an equation to the system to determine the particle velocity. The literature [Maxey and Riley, 1983] gives a second equation for each particles velocity component :

$$\frac{d}{dt}\boldsymbol{x} = \boldsymbol{v} \tag{32a}$$

$$\frac{d}{dt}\boldsymbol{v} = -(\boldsymbol{v} - \boldsymbol{u})/\tau_p \tag{32b}$$

$$\boldsymbol{v}(\cdot, 0) = \boldsymbol{u}(\cdot, 0) \tag{32c}$$

where $\boldsymbol{v} := (v_1, v_2)$ is the velocity of the particle, $\boldsymbol{u} := (u_1, u_2)$ is the velocity of the fluid and $\tau_p$ represents the relaxation time for the inertial particles As in the fluid particle case, the initial positions are uniformly distributed within the square $[0, 2\pi]^2 \in \mathbb{R}^2$ and then injected into the fully developed flow to be tracked in the Lagrangian framework.

As shown by the equation (32c), in our model the initial velocity of the particles is the same as that of the fluid. The model of Maxey-Riley is based on two assumptions : Firstly, the model is for Stokes particles, i.e. spherically shaped particles having a small particle Reynolds numbers. The ratio of the particle density to the fluid is larger than 1. Assuming a finite viscosity $\nu > 0$, the second condition appears in the definition of $\tau_p$ :

$$\tau_p = \frac{\rho_p}{\rho_f}\frac{2r_p^2}{9\nu} \tag{33}$$

with $\rho_p$ the particle density, $\rho_f$ being the fluid density, $r_p$ represent the radius of the particle and $\nu$ is the kinematic viscosity. Given these notations, the second condition becomes :

$$\frac{\rho_p}{\rho_f} >> 1$$

Maxey's point particle model [Maxey and Riley, 1983] with Stokes drag is used and the inertial dynamics is controlled by the Stokes number, $St = \tau_p/\tau_\eta$ where $\tau_p$ is the particle relaxation time and $\tau_\eta$ the Kolmogorov time, the latter being defined as :

$$\tau_n = \left(\frac{\nu}{\epsilon}\right)^{\frac{1}{2}}$$

where $\epsilon = -dE/dt$ represents the kinetic energy dissipation.

This is a one-way coupling where the fluid velocity drives the particle motion, but there is no interaction of the particles with the fluids. Furthermore, there is no interaction between particles such as collision.

For the second equation, we implement a Runge-Kutta approximation of second order, i.e. :

$$\boldsymbol{v}_{n+1} = \boldsymbol{v}_n + \frac{1}{2}\left(k_1^{(n)} + k_2^{(n)}\right)$$

By defining $f(t, \boldsymbol{v}) := -(\boldsymbol{v} - \boldsymbol{u})/\tau_p = \frac{d}{dt}\boldsymbol{v}$, the Runge-Kutta method gives us for $k_1^n$, $k_2^n$ at time $t_n = n\Delta t$ :

$$k_1^n = \Delta t f(t_n, v_n), \qquad k_2^n = \Delta t f\left(t_n + \frac{\Delta t}{2}, v_n + \frac{k_1^n}{2}\right)$$

An explicit Euler scheme is used for equation (32a)[3] :

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta t \boldsymbol{v}^n \tag{34}$$

The code simultaneously run 21 simulations with batch of $1/2$ millions particles for different Stokes numbers[4]. One of them is $\tau_p = 0$ which corresponds to fluid particles, this represents a total of $21 \times 0.5 \times 10^6 = 10.5$ millions particles advected and it terms of memory it takes roughly 168 MB[5] on the GPU memory without any data compression.

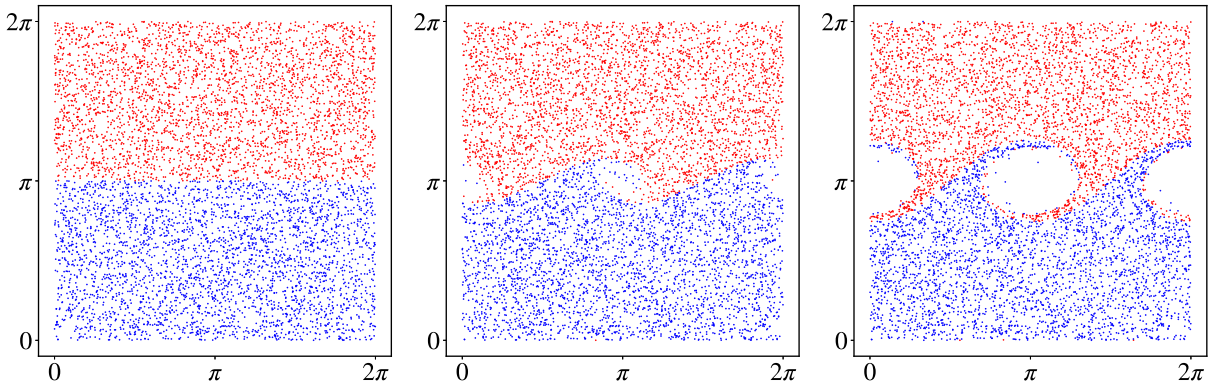
In the same fashion as the fluid particles and using the same parameters, we can look at the particle advection for the single shear layer condition with inertial particles, in the case below we take $\tau_p = 1$.

As seen in equation (32b), the advection of an inertial particle depends not only on the velocity of the flow but also on by the value of $\tau_p$. In this case the mass of the particle increases with the value of $\tau_p$. We saw in figure 6 the case of fluid particles which corresponds to $\tau_p = 0$. In figure 8, the advection of the system (32) is done with $\tau_p = 1$. The main difference can be seen at time $t = 50$ where we can nicely observe the particles avoiding high-vorticity regions at the interface between the two layers due to centrifugal effects which eject the particles from vortical regions.

[3] This can be changed to a full RK2/RK3 by saving the previous particles velocity into an Hermitian array

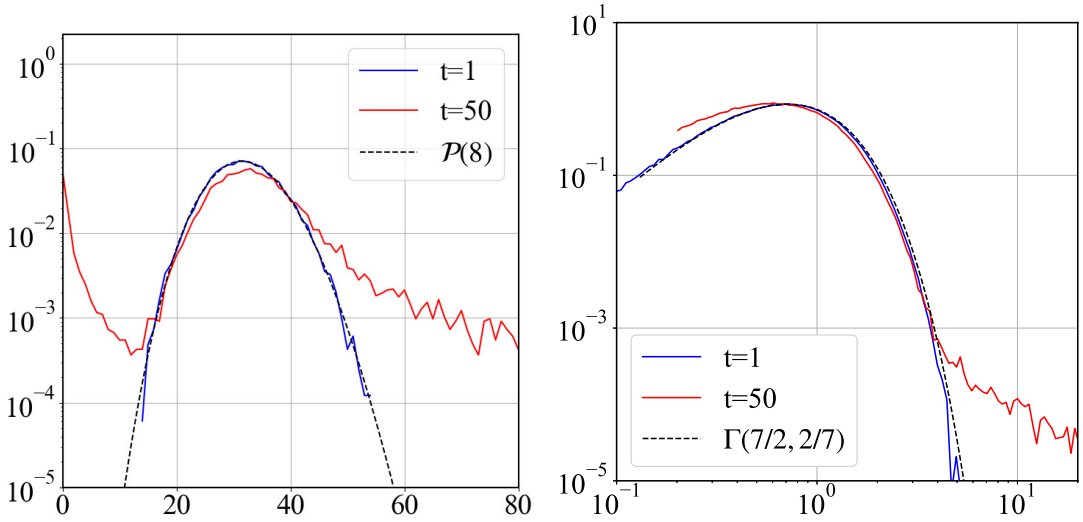
[4] the exact values used are $\tau_p \in \{0, 0.005, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.125, 0.15, 0.25, 0.5, 0.75, 1, 2, 5, 13\}$

[5] For each $\tau_p$, the particles are stored in a unidimensional array as double precision numbers




(a) $t = 1$ (b) $t = 35$ (c) $t = 50$

Figure 8: Shear layer inertial particle advection with $\tau_p = 1$ on a $128^2$ coarse grid at time $t = 1$, 35 and $t = 50$.




(a) Density (b) Voronoi

Figure 9: PDF of the density and the Voronoi volume computed for inertial particles($\tau_p = 1$) on the mixing layer condition at time $t = 1$ and $t = 50$. The dashed lines represent the Poisson (left) and the Gamma (right) distribution.

Figure 9 shows at time $t = 1$ and time $t_{final} = 50$ the PDF of the density of inertial particles computed with bins of size $(2\pi/128)^2$. The dashed line represents the Poisson distribution. We can observe that at time $t = 1$ the PDF is superimposed with the theoretical PDF of randomly distributed particles, but at time $t_{final} = 50$ the particle density no longer follows the Poisson distribution. There are more dense and void regions as it can be observed in figure 8.

# 5 Numericals Results

## 5.1 Studied flows

In this study, we look in more details on four initial condition for the vorticity : Mixing layer (single shear layer), 4-mode flow, Two-dimensional turbulence starting with gaussian blobs and Two co-rotating vortices.

- Mixing layer (single shear layer)
- 4-mode flow
- Two-dimensional turbulence starting with Gaussian blobs
- Two co-rotating vortices

This section explains these conditions, as well as a visualization and the equation used to produce the initial vorticity. The color map used for the vorticity plots was developed in Farge [2000].

### 5.1.1 Single shear layer

The single shear layer, also called the mixing layer, is a temporally developing mixing layer. We have two horizontal velocities : on the top half from left to right and on the bottom half from right to left (or vice-versa) which produce a shear flow and thus generate vorticity at their interface. The flow is unstable whether it is inviscid or not, it produces an instability called Kelvin-Helmholtz instability and the vorticity line starts to oscillate and rolls up into vortices called Kelvin-Helmholtz vortices (see Michalke [1972]).
The equation generating the initial vorticity $\omega_0$ for the mixing layer condition is defined as follows :

$$\omega_0(x, y) = (1 + \Delta_1 \cos(2x)) \exp(-\Delta_2(y - \pi)^2)$$

where $\Delta_1 = 0.01$ introduces a small perturbation in the shear layer and $\Delta_2 = 50$ controls the thickness of the shear.

In figure 10, we can visualize the vorticity field of the mixing layer for a $1024^2$ coarse grid, $4096^2$ fine grid and an incompressibility threshold of $10^{-2}$. We observe in (top, left), the shear seems to split the domain into equal parts, but the $\Delta_1$ quantity add an asymmetry that will produce the instability and the formation of vortices. At time $t = 50$ (top, right) we observe a fully developed Kelvin-Helmholtz instability. At $t = 200$ (bottom, left), the Kelvin-Helmholtz vortex wrap around itself developing more fine scale.
Furthermore we can nicely observe a zoom (bottom, right) on the vortex at time $t = 200$. This zoom is made using the property of the CM method , i.e. the ability to split the map into a composition of sub-maps which allow us to see much finer scale, than the grid the computation was made on. This process is also closely related to the remapping procedure explained earlier.

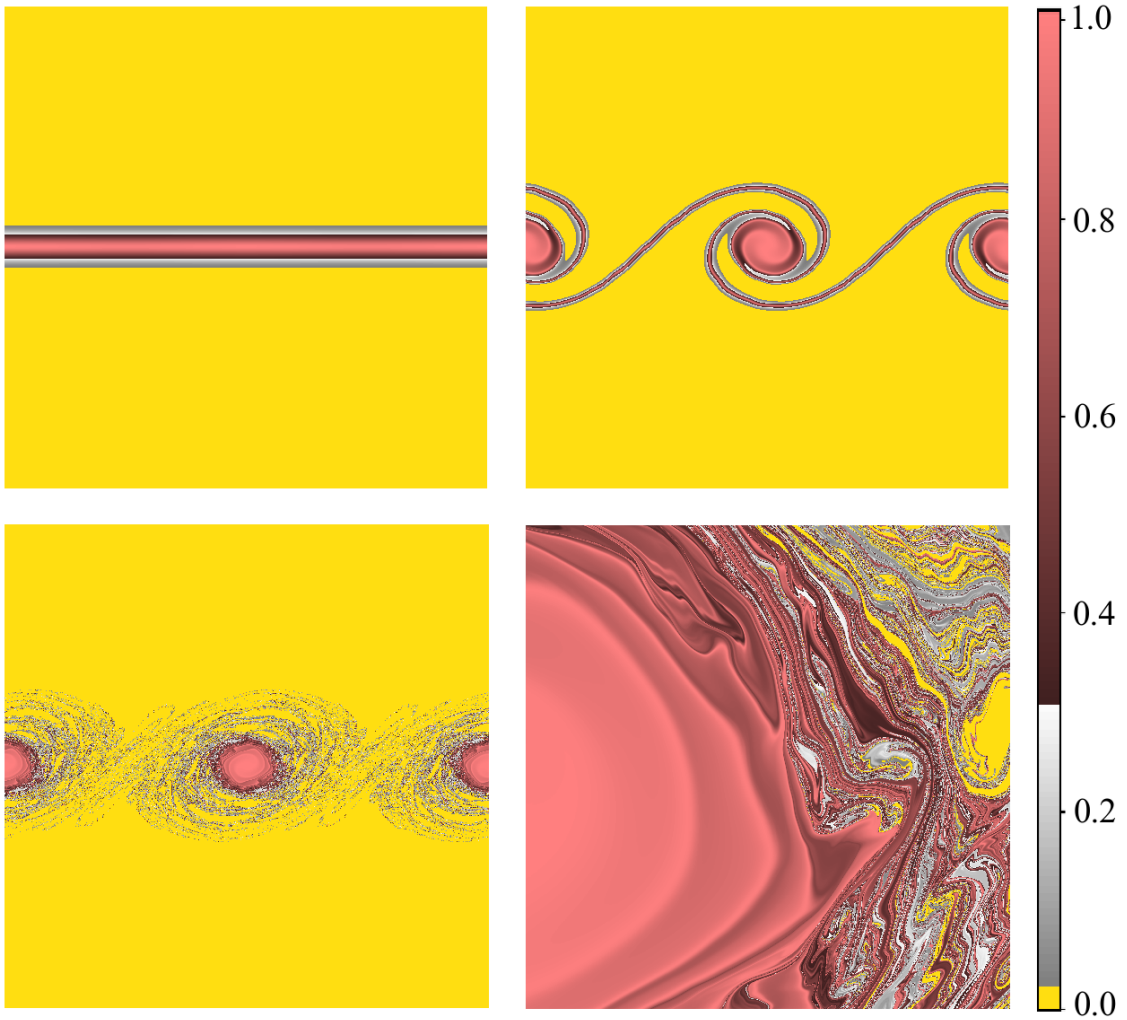



Figure 10: Vorticity of the mixing layer at $t = 0$, $t = 50$, $t = 200$ and a zoom on the vortex at time $t = 200$ using the same principle as remapping.

### 5.1.2 4-mode flow

The 4-mode flow is a simple test case which can be found in Frisch [1995] for validating the Cauchy-Lagrange method. This flow is often used to compare results obtained with different numerical methods, as it develops high vorticity gradients in short time. The initial condition only contains 4 Fourier coefficients. The generating equation for the initial vorticity is :

$$\omega_0(x, y) = \cos(x) + \cos(y) + 0.6\cos(2x) + 0.2\cos(3x)$$

We can visualize it :

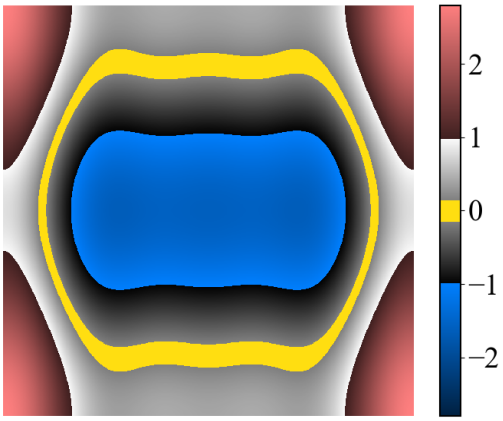



Figure 11: Vorticity at $t = 0$ for the 4-mode flow.

Figure 11 shows the initial vorticity for the 4-mode flow, we can observe one counter-clockwise vortex (blue) at the center and four clockwise vortices in the corner (in red).

We can compute the exact initial enstrophy $Z(t = 0)$. This will be helpful for the plot over time of $Z$. By developing the square and taking into account that $\cos(nx)_{n\geq 0}$ is an orthogonal basis, we can simplify the computation and deduce by hand :

$$Z(t = 0) = \frac{60}{25}\pi^2 \approx 23.6870505626$$

In a similar fashion, we can also compute the initial kinetic energy $E(t = 0)$ of the flow. By using (6) and an integration by part, we can deduce the initial energy from this formula :

$$\int_{\mathbb{T}^2} |\boldsymbol{u}_0|^2 d\boldsymbol{x} = -\int_{\mathbb{T}^2} \omega_0 \psi_0 d\boldsymbol{x}$$

We still need to find the expression of $\psi_0$ which can also be simply done by integrating twice in the $x$ and $y$ direction, since $\omega_0$ has both variables separated, i.e. $\omega_0 = f(x) + g(y)$ for some function $f, g$ that can be easily deduced from the expression of $\omega_0$. Hence we have by hand:

$$\psi_0(x, y) = -\left(\cos(x) + \cos(y) + \frac{3}{20}\cos(2x) + \frac{1}{45}\cos(3x)\right)$$

From here, we deduce the initial energy :

$$E(t = 0) = \frac{377}{180}\pi^2 \approx 20.67133810672604$$

These computations will be useful later on when plotting the energy and enstrophy over time for the 4-mode flow condition.

### 5.1.3 Two-dimensional turbulence

The initial condition of the two-dimensional turbulence is set in a $2\pi$-periodic domain with a total of 98 Gaussian blobs with the same number of positive and negative vortices, equidistantly distributed with a variance $\sigma = 0.2$. The symmetry is broken with a random perturbation in the position of the negative blobs.
Due to the periodicity of the domain and the disposition of the Gaussian, the average vorticity is however found to be non-zero. We modify this value by setting the first value of the power spectrum to zero in order not to be in a domain with a global rotation. Furthermore, we add a cut-off scale on Fourier transform of the vorticity on the coarse grid at $2/3$ of frequency, i.e. a dissipation scale, so that the turbulence does not transform into Gaussian noise.
The enstrophy spectrum is defined correspondingly via the vorticity, $\omega$:

The numerical parameters are a coarse grid of resolution of $2048^2$, a fine grid resolution of $16384^2$. For time discretisation we use step size $\Delta t = 2.5 \times 10^{-4}$ and an incompressibility threshold $\delta = 3 \times 10^{-3}$. The value of the incompressibility threshold has been chosen so that there is not too much remapping and to save in memory before the turbulence appears.

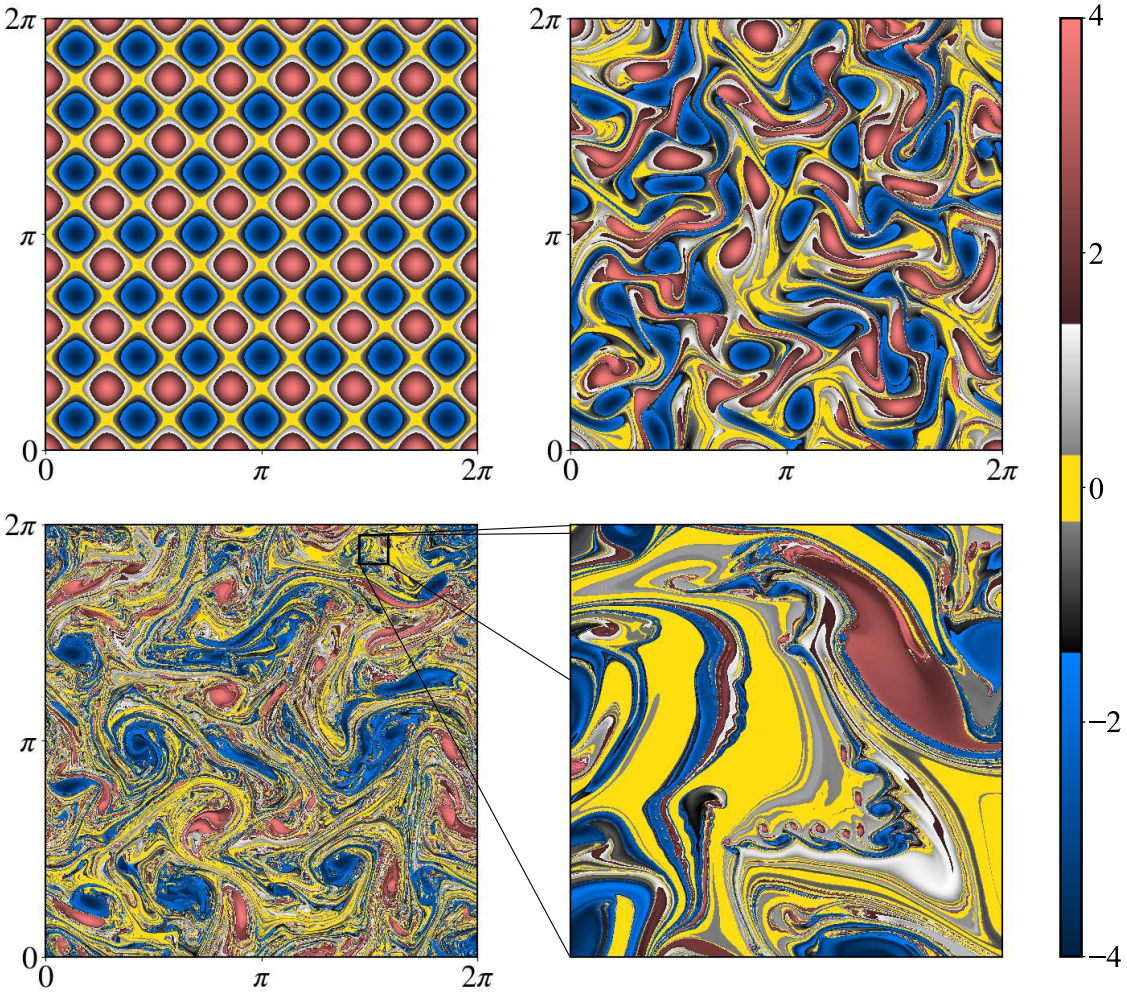



Figure 12: Vorticity of two-dimensional turbulence at time $t = 0$ (top, left), 15 (top, right), 30 (bottom, left) and a zoom at $t = 30$ (bottom, right) with a window of size $2\pi/16$.

Figure 12 shows the evolution of the vorticity of two-dimensional turbulence at time $t = 0$, 15, 30 and a zoom at $t = 30$. At first the different blobs are distributed equidistantly, then due to

the small asymmetry we introduced, the blobs move and are slightly deformed. Then we can observe the presence of filaments and coherent vortices at small scale, as we can see on the figure 12 (bottom, right) which is a 16 times zoom on the domain.

#### 5.1.4 Two co-rotating vortices

In the two co-rotating case, there are two possible outcomes depending on the viscosity term $\nu$ : In the Navier-Stokes case, we have that the two vortices merge by first rotating around each other, going into a shear and then merging into a single larger vortex due to viscous effect. In the Euler case, the vortices wrap around each other generating finer and finer scales up until a point where the simulation cannot be reversed as the scales are not resolved anymore. Formally the initial vorticity field is generated by superimposing two Gaussian functions centered at two different positions :

$$f(x,y) = \sin(x)^2 \sin(y)^2 \Bigg( \exp\left(5(x-\pi)^2 + \left(y - \frac{\pi}{3}\right)^2\right) + \exp\left(5(x-\pi)^2 + \left(y - \frac{2\pi}{3}\right)\right)^2 \Bigg)$$

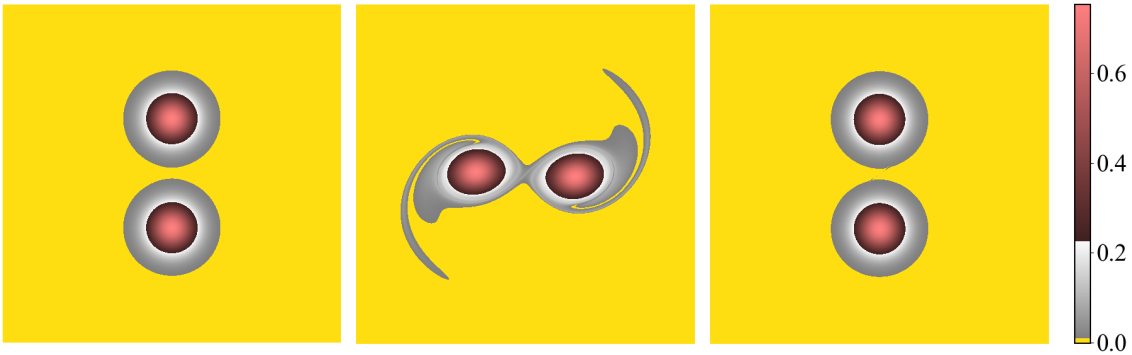



Figure 13: Vorticity for the two co-rotating vortices on a $128^2$ coarse grid and a $2048^2$ fine grid at $t = 0$, $t = 60$ and $t = 120$ with the time being reversed at $t = 60$.

Since the Euler equations are time reversible, we can take the simulation to a certain point and from here, we can choose a negative time step allowing us to compute the inverse simulation. This process can be done up until the moment where eddies take shape after the shear resulting from the fusion of the two vortices. At this moment small and large scale feature intertwine and back pedaling the simulation does not give the expected result.

To test the reversibility of CMM, a simulation has been performed and time has been inverted at a given time instant. Figure 13 shows the vorticity field of two co-rotating vortices at time $t = 0$, $t = 60$ and the solution if we invert the time step at $t = 60$. We observe that we can retrieve the initial condition almost perfectly even if a filament is developed between the two vortices.

If we reverse the time for the inertial particles, their velocity increases exponentially. That is to be expected since equation (32) is not time reversible with the introduction of the coefficient $\tau_p$ which is the factor that introduces inertia.

## 5.2 Code validation

In this section, we validate the implemented CUDA code of the CM method using the 4-mode flow. The obtained results are compared with those obtained with the MATLAB code used in Yin et al. [2021a] and the results obtained with a Cauchy Lagrange method (which were confronted with the results of a high resolution spectral method) by Podvigina et al. [2016]. CPU times and GPU times together with memory requirements are reported and confronted with those obtained with the MATLAB code. The convergence properties of the method in space and time are assessed numerically and compared with the theoretical results provided in Yin et al. [2021a]. In addition to that, the conservation properties of energy and enstrophy are analyzed.

Except for the convergence test below, the simulations were made on a $128^2$ coarse grid and a $512^2$ fine grid with a time step $\Delta t = 1/32$ and an incompressibility threshold of $\delta = 10^{-4}$. The span of the simulation is $8/\Delta t$ = 256 time steps.

### 5.2.1 Time evolution of energy, enstrophy and palinstrophy

First we look at the time evolution of the energy, enstrophy and palinstrophy. One important property of the numerical method to be checked is the conservation of the energy and the enstrophy which should be satisfied in the case of the Euler equations.

Figure 14 shows the evolution of the energy, enstrophy and palinstrophy on different grids. The computations of the three quantities are done in real space and use for quadrature a Riemann sum in 2D, hence the coarser the grid the less accurate is the result. Since $\psi$ is computed on the coarse grid, the energy also needs to be computed on the $128^2$ grid while the enstrophy and the palinstrophy can be computed on the fine grid of resolution $512^2$. Nevertheless, we can observe that the initial energy, enstrophy are in agreement with the previously computed theoretical values in section 5.1 and as for the conservation side, we observe a maximum difference with the initial values of the energy and the enstrophy in the order of $6 \times 10^{-3}$ and $2 \times 10^{-4}$, respectively.

As for the palinstrophy, we can see an exponential increase due to the increase in vorticity gradients. This can be seen at the interface between the centered counter-clockwise vortices and the four clockwise vortices at the corner of the domain. These small scale features can be seen as well in the isolines shown further down.

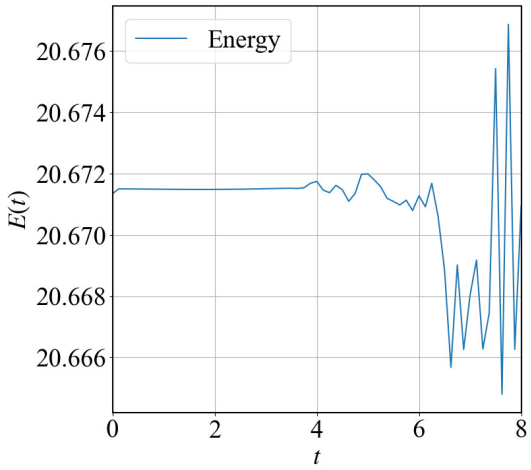
### 5.2.2 Enstrophy spectrum

Similar to Yin et al. [2021a], we plot the enstrophy spectrum obtained from the simulations carried out with a $128^2$ coarse grid which was then upsampled to a $1024^2$ grid in order to see behaviour at high wave numbers. Alongside the CMM data (orange), we plot the data from Podvigina et al. [2016] computed on a $8192^2$ grid using a method called the Cauchy-Lagrangian method which is more detailed in the aforementioned paper.

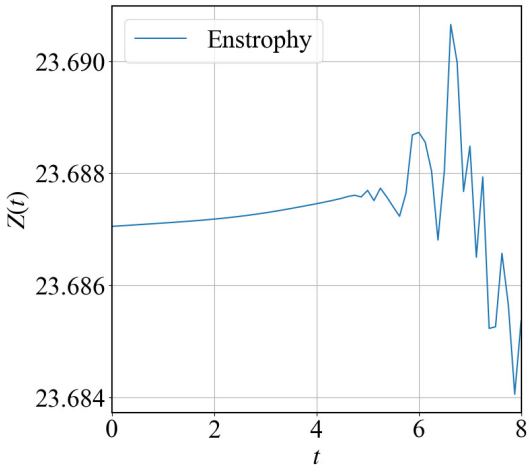
.

Figure 15 shows the enstrophy spectrum at time $t \approx 3.5$ and $t \approx 4$ using the data from the CUDA code (orange) and the data from Podvigina et al. [2016]. Analyzing these spectra, we can see that all wave numbers are well preserved, except the high wave numbers, when the enstrophy spectrum has values below $10^{-12}$. For these the CMM results show a slower decay than the Cauchy-Lagrange method (CL8).

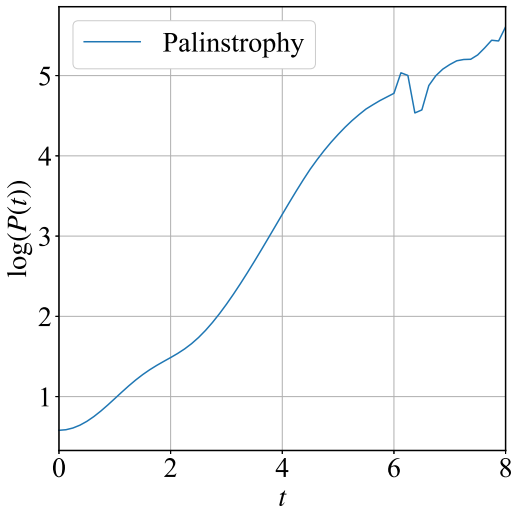
[6]Since our time step is equal to $\Delta t = 1/32$, we tried to plot the spectra at time as close as possible to the one in the referenced data which are $t = 3.45$ and $t = 3.95$. The exact time in our case are respectively $t = 110\Delta t = 3.4375$ and $t = 126\Delta t = 3.9375$




(a) Energy on the coarse grid.




(b) Enstrophy on the fine grid.



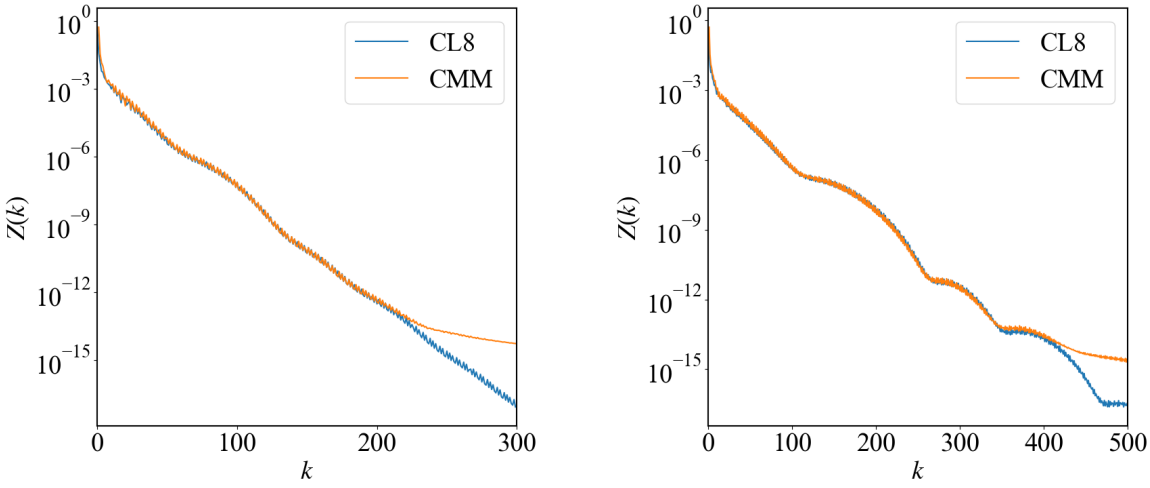

(c) Palinstrophy on the fine grid.

Figure 14: Evolution of the energy, enstrophy, palinstrophy over time for the 4-mode flow condition on a $128^2$ coarse grid for the energy and a $512^2$ fine grid for the rest.

The CM method give almost the same results as CL8, while having a resolution which is 63 times coarser. The computed enstrophy spectrum deviates sooner from the CL8 data and the spectra obtained with the MATLAB code. The most plausible hypothesis for this problem might come from the order of $\epsilon$ used in the CUDA code for the map advection. Since the CUDA code uses a second order finite difference method for the map advection. We have to take in our computations small values of $\epsilon = 10^{-6}$. Given that we advect in equation (26) the cross derivatives which implies a multiplication by $1/\epsilon^2$ using a finite difference scheme, implies that in this case we have the order $10^{-12}$. Taking into account that our machine precision is of the order $10^{-16}$, this let us with a small room of maneuver for the map cross derivative order, i.e. :

$$4\epsilon^2 \partial^2_{xy}\chi(\boldsymbol{x}, t_{n+1}) > 10^{-16} \implies \partial^2_{xy}\chi(\boldsymbol{x}, t_{n+1}) > \frac{1}{4\epsilon^2}10^{-16} = 2.5 \times 10^{-5}$$

If the above inequality does not hold, then we loose some decimal precision for the last equation of (26). The same inequality appears for the $x$ and $y$ derivatives of the map $\chi$ even though




(a) $t \approx 3.5$ (b) $t \approx 4$

Figure 15: Enstrophy spectrum of the 4-mode flow condition at time[6] $t = 3.5$ and 4. Computations for CMM on a $128^2$ grid which was upsampled to $1024^2$. CL8 denotes results obtained with the Cauchy-Lagrange method from Podvigina et al. [2016].

it is less restrictive :

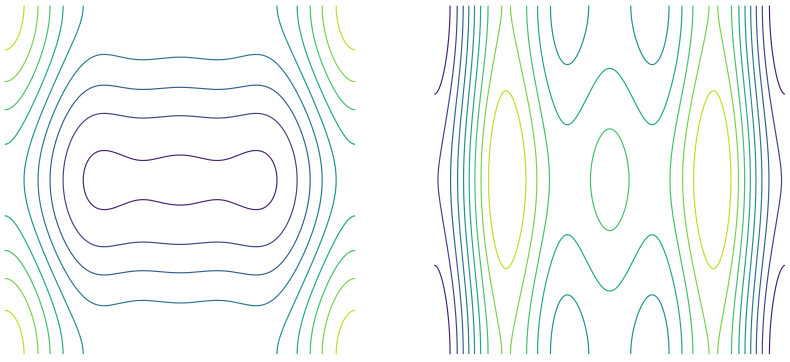
$$\partial_x \chi(x, t_{n+1}) > 10^{-10}, \qquad \partial_y \chi(x, t_{n+1}) > 10^{-10}$$

Further test on the range of derivatives of the map $\chi$ in the case of the 4-mode flow needs to be done in order confirm the affirmation above.

### 5.2.3 Flow visualization

The initial condition of the vorticity and its Laplacian (later used to illustrate the small scale features of the flow) can be seen on figure 16. For the 4-mode flow condition, the vorticity value are in the interval $[-2, 3]$, we took nine evenly spaced contour lines with the following values :

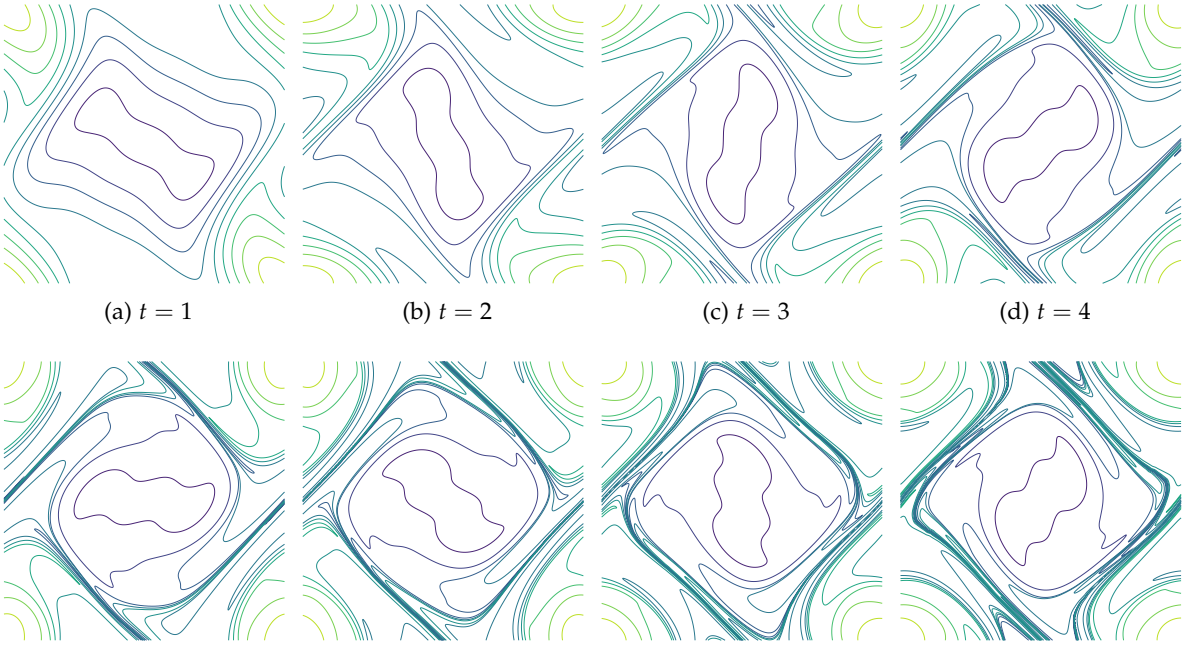
$$[-2, -1.5, -1, -0.5, 0, 0.5, 1, 1.5, 2, 2.5, 3]$$



(a) $\omega_0$ (b) $\Delta\omega_0$

Figure 16: Contour plot for the initial vorticity and its Laplacian on the 4-mode flow.

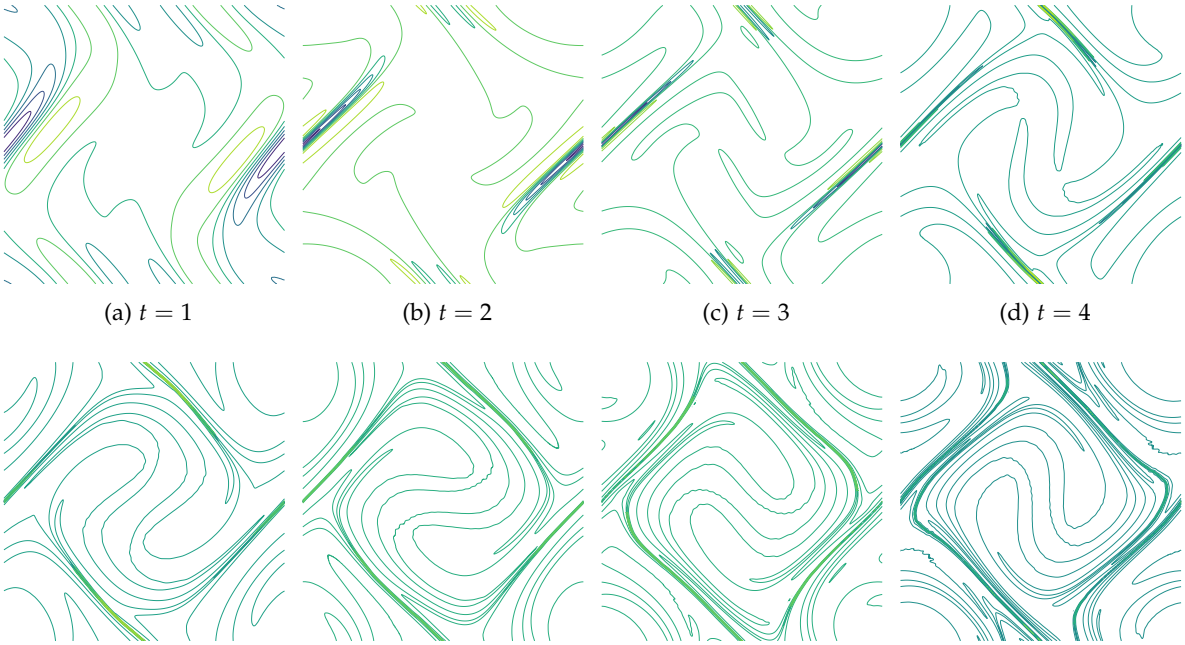
The vorticity isolines are in agreement with the one in Yin et al. [2021a]. In figure 16, we observe the large vortices at the center and the corner of the periodic domain which rotate in opposite direction. From $t = 5$, we can see finer features appearing in the simulation at the interface between the large centered vortex and the four smaller vortices. Moreover, in figure 18, we can similarly observe in between the main vortices the accumulation of fine structures that are more apparent starting from $t = 3$. Although the map $\chi$ which is a composition of sub-maps evolved on a $128^2$ coarse grid, we can see that the vorticity develops much finer scales that were not present in the initial condition.



(a) $t = 1$ (b) $t = 2$ (c) $t = 3$ (d) $t = 4$

(e) $t = 5$ (f) $t = 6$ (g) $t = 7$ (h) $t = 8$

Figure 17: Contour plot of the vorticity $\omega$ for the 4-mode flow condition on a $128^2$ coarse grid (upsampled on a $1024^2$ grid).



(a) $t = 1$ (b) $t = 2$ (c) $t = 3$ (d) $t = 4$

(e) $t = 5$ (f) $t = 6$ (g) $t = 7$ (h) $t = 8$

Figure 18: Contour plot of the Laplacian of the vorticity $\Delta\omega$ for the 4-mode flow condition on a $128^2$ coarse grid (upsampled on a $1024^2$ grid).

#### 5.2.4 CPU/GPU time comparison

The accuracy of the results between the CUDA and MATLAB code and also CL8 [Podvigina et al., 2016] has been compared above . The implementation of the CUDA and MATLAB code differs, nevertheless we are able to reproduce the MATLAB results published in Yin et al. [2021a] to a sufficient degree. In the following, the execution time of the CUDA code on two different graphics card machines will be compared to the MATLAB execution code. As we did not have access to the MATLAB code we could not directly run it with our computational resources and the absolute run time cannot be compared. For this reason we also consider the number of remappings.

For each table below, we run the same 4-mode test with the same parameters. We compare the MATLAB code to the original CUDA code provided by B. Yadav using a second order Adams-Bashforth implementation (AB2) for the time integration. The updated CUDA code with third order Runge-Kutta time integration (RK3) is likewise considered. Computations are done on three different machines : a laptop computer and two supercomputers (Mésocentre of Aix-Marseille Université and Anticythere of I2M), where the latter are more common in the field of high performance computing.

The simulation for the following table were carried in MATLAB[7] with a computer equipped with an Intel Core i5-2320 3.00GHz 4 cores processor and 8GB of RAM :

| t | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Number of remaps | 1 | 4 | 14 | 30 | 47 | 65 | 88 | 105 |
| Total of CPU time (in sec) | 20s | 41s | 66s | 100s | 145s | 202s | 274s | 359s |

The following benchmark test was made on a laptop equipped with an Intel Core i7-10750H 2.60GHz 6 cores processor, 16GB of RAM and a RTX 3060 GPU with 6GB of video mem-

[7] See table 1 from Yin et al. [2021a]

ory (VRAM). For that a second order Adams-Bashforth scheme coupled with a fixed point iteration for the time integration is used :

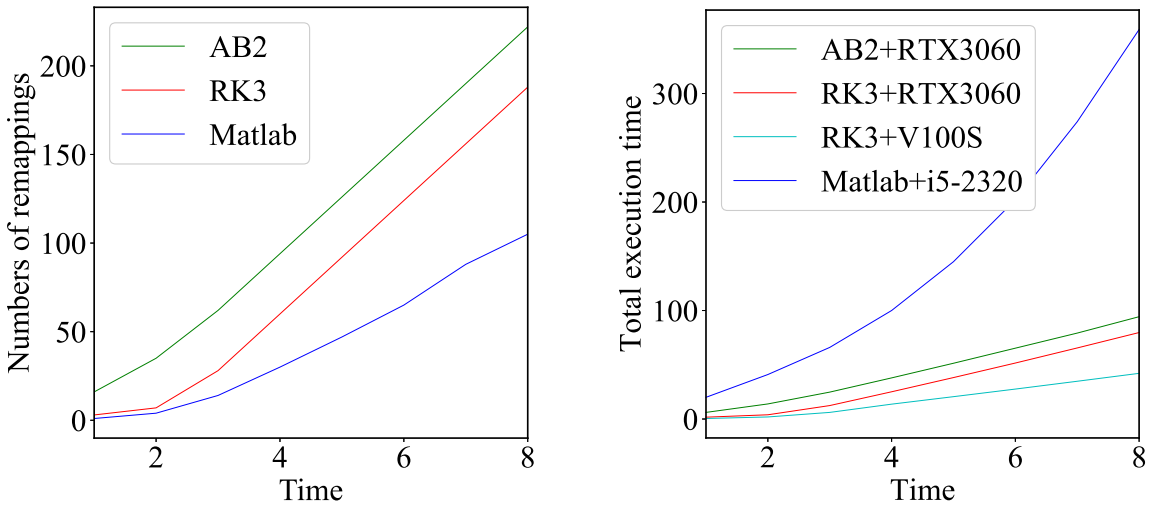
| t | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Number of remaps | 16 | 35 | 62 | 94 | 126 | 158 | 190 | 222 |
| Total execution time (in sec) | 6.10s | 13.9s | 24.8s | 37.9s | 51.4s | 65.3s | 79.2s | 94.3s |

We can see that the original Cuda code with the multi-step Adams-Bashforth method improves the speed despite the large number of remappings, which far exceeds those of the MATLAB code at each considered time instant. Using the same laptop, we also tested the third order Runge-Kutta time integration with Lagrange time interpolation:

| t | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Number of remaps | 3 | 7 | 28 | 60 | 92 | 124 | 156 | 188 |
| Total execution time (in sec) | 1.7s | 3.9s | 12.4s | 25.1s | 38.2s | 51.6s | 65.5s | 79.7s |

We also test the RK3 time integration with a new machine at I2M (Anticythere) equipped with an Intel Xeon Bronze 3204 1.90 GHz 6 cores processor, 196 GB of RAM and a NVIDIA Tesla V100S GPU with 32 GB of VRAM.

| t | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Number of remaps | 1 | 6 | 25 | 57 | 89 | 121 | 153 | 185 |
| Total execution time (in sec) | 0.48s | 1.9s | 6.1s | 13.7s | 20.6s | 27.6s | 34.8s | 42.1s |




(a) Remapping over time

(b) Total execution time over time

Figure 19: Number of remappings over time comparing the CUDA codes with AB2 and RK3 and the Matlab code (left) and total execution time over time instant on different machines for the 4-mode flow condition (benchmark).

Remapping is a costly process in terms of computation. From the results we can see that the implementation of RK3 further reduces the number of remappings, as shown in figure 19a. The amount of remappings done are an important variable and show the optimization of the code.

On 19a (left), we can see that the slope for the AB2 and RK3 methods are the same after a certain amount of time ($t = 2$). The slope of the MATLAB code is roughly 18, while the AB2 and RK3 method in the CUDA code have a slope of approximatively 32. This difference in

scaling might be due to some implementation lacking on the CUDA code such as for example the missing mollification and upsampling from a grid resolution of $128^2$ to $512^2$ of the "Hermitian" array $\psi$ which contains the values of the stream function $\psi$ and its derivatives $\partial_x\psi, \partial_y\psi, \partial_x\partial_y\psi$. Thus the CUDA code uses a less smoother $\psi$. There is also the map advection presented in equation (26) which in our case employs a second order centered finite difference scheme, whereas the MATLAB code uses a fourth order method allowing an important decrease in the order of $\epsilon$. In the meantime it also maintains a certain distance from the machine precision. The remapping memory footprint has been a limiting factor for the CUDA code at one point - indeed all the remappings need to be saved in a stack in order to reinitialize the map. There is a problem for high resolution grids and low incompressibility thresholds, since the GPUs in our hand have at best only 32 GB of VRAM and taking into account that before the improvement, the CUDA code had to do a remapping every time step, the memory was often saturated. Besides the decrease of remapping at the beginning of the simulation (c.f 19a), we had to sacrifice some speed for high resolution grid/small incompressibility threshold by putting the remapping stack into the RAM instead of solely keeping it on the VRAM. In terms of speed, we traded the time it takes us to do a computation into a transfer from the GPU to the RAM and central processing unit.

Figure 19b shows the total execution time over the simulation time save of the different machine and implementation used. We can see that despite the number of remappings, which is higher than for the MATLAB code, the CUDA implementation is overall faster. The time complexity for the MATLAB code seems to be in $O(t^2)$ and as for the CUDA code, a more lenghty simulation should lead to the same complexity. The memory complexity for the remapping process is linear since we save as many maps as needed.

### 5.3 Convergence tests

To make numerical convergence tests in space and time, we use the 4-mode flow condition and the same parameters as Yin et al. [2021a]. We take for reference a simulation on a fine grid with resolution $1024^2$ i.e. $\Delta x = 1/1024$ with a time step $\Delta t = 1/512$. The vorticity is sampled on a $1024^2$ fine grid. In order to see the convergence errors in the time variable, we fix $\Delta x = 1/1024$ and go through different time-steps $\Delta t \in \{1/32, 1/64, 1/128, 1/256, 1/512\}$. The results are done without remapping and we compute the error of the different quantities based on the following norms :

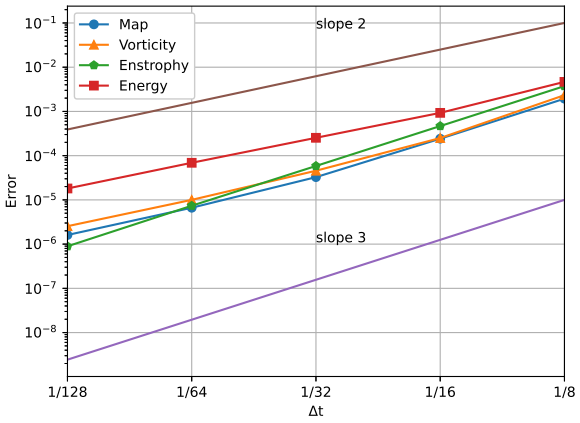
$$\text{Map error} := ||\chi^n - \boldsymbol{X}(\cdot, t_n)||_\infty \tag{35a}$$

$$\text{Vorticity error} := ||\omega^n - \omega(\cdot, t_n)||_\infty \tag{35b}$$

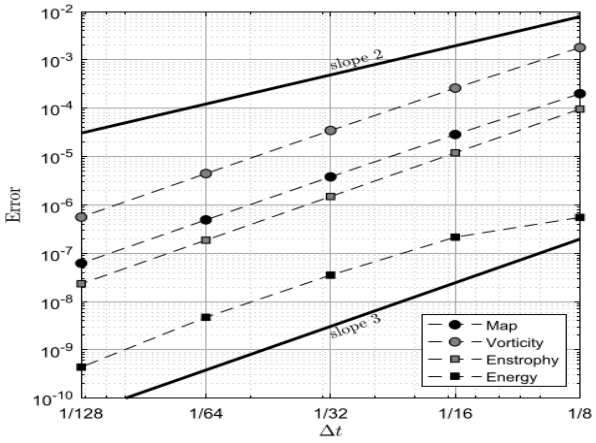
$$\text{Enstrophy conservation error} := ||\omega^n||_2^2 - ||\omega_0||_2^2 \tag{35c}$$

$$\text{Energy conservation error} := ||u^n||_2^2 - ||u_0||_2^2 \tag{35d}$$

While due to discrepancies between the MATLAB and CUDA code the magnitude of the precision differs, comparing the slopes is the key point, as it represents the order of convergence: Figure 20 shows the errors defined in (35) as function of the time step $\Delta t$. We observe a third order error for the enstrophy, a slope of 2.6 for the flow map error, a slope of 2.4 for the vorticity error and a slope of 2.0 for the energy error. These slopes are expected, except for the energy cerror which is of order 2 and not 3, as observed in Yin et al. [2021a]. This might come from the fact that the stream function $\psi$ is on a coarser grid than the vorticity and also the initialization of previous values of $\psi$ where $\psi_p$ is computed and we set $\psi_{pp} = \psi_p$, which should reduce to an explicit Euler method for the initial step.



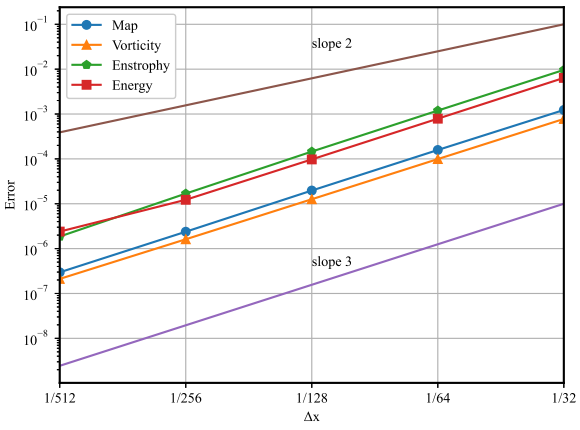

(a) CUDA time convergence



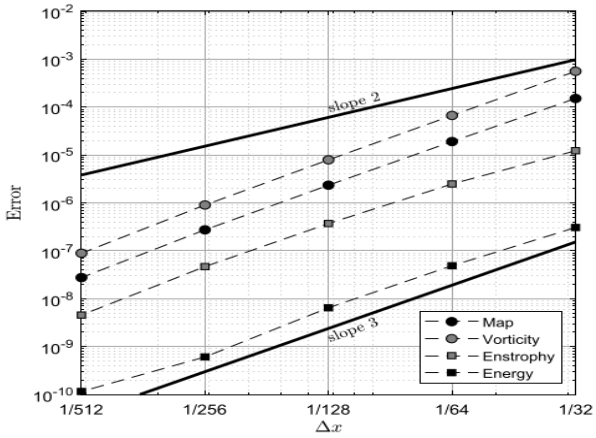

(b) MATLAB time convergence(Yin et al. [2021a])

Figure 20: Time convergence for the 4-mode flow: errors as a function of the time step without remapping.

For the spatial error, we take $\Delta t = 1/512$ fixed, while $\Delta x$ is varying through the values in the following set : $\{1/32, 1/64, 1/128, 1/256, 1/512\}$ :




(a) CUDA spatial convergence




(b) MATLAB spatial convergence(Yin et al. [2021a]) here

Figure 21: Space convergence for the 4-mode flow: errors for the spatial step without remapping.

Figure 21 shows the errors defined in (35) as function of the space discretisation step $\Delta x$. We observe a third order convergence for the map error, the vorticity error, the enstrophy and energy errors. This is expected from the Hermite cubic interpolation used, which is of order $O(\Delta x^3)$. For detailed numerical analysis of time and space errors showing global third order accuracy we refer to Yin et al. [2021a].

# 6 Conclusions and perspectives

The Characteristic Mapping method is a semi-Lagrangian formulation which allows us to solve non-linear advection problems, in the present case the incompressible 2D Euler equations, with rather low computational cost while preserving the fine scale features of the solution and high resolution using coarse grids. The flow map is evolved using a gradient augmented level set method and FFT techniques are used for efficient reconstruction of the velocity field. Remapping allows error control of the incompressibility condition. Computational efficiency and the high precision of CMM were analyzed optimizing an existing Cuda code, which is implemented and adapted on recent graphic cards. Different validation tests were performed illustrating third order accuracy.

The properties of the flow map, in particular the decomposition into sub maps and the remapping are the reasons behind the possibility of the zoom capability of the method. Moreover, the CM method has proven to be of great affinity with parallelization thanks to its interpolation compactness (cubic Hermite polynomials). The CM method also proves to be robust indeed, since it solves non linear advection equations in a Lagrangian reference frame. There are multiple systems of equation falling under such premise, one of them being the Vlasov-Poisson equation which from a Lagrangian point of view splits the problem into two transport equations (for the velocity and position) and a Poisson equation for the magnetic field. Hence it is quite straight forward to expect an application to those types of problems. During this internship, we updated the code readability and its ability to run it on modern CUDA toolkits. We have added a third order Lagrange interpolation for the time-scheme and modified the time integration of the flow map from a second order Adam-Bashforth/Moulton method to a third order Runge-Kutta scheme. We showed that this modification was partially reducing the costly process of remapping. In order to study Lagrangian turbulence, a point particle model for inertial particles was introduced into the code: The Maxey-Riley model can be used with different Stokes number in the code. These fluid/inertial particles run with a second order Runge-Kutta scheme even though a convergence plot of the error remains to be done to prove the order of the method numerically. The CUDA code is still a work in progress but it already produces reliable results. Moreover, there is some potential such as a simple upgrade of the map advection from second order centered scheme to a fourth order finite difference scheme and the particles which could use a third order Runge-Kutta scheme without implying too many drawbacks to the computation. Finally, there is the potential for a multiple GPU implementation. The 2D implementation of the CM method along with particle advection in CUDA is a test case to see how reliable is the method for 2D. A foreseeable CUDA implementation would be for the incompressible 3D Euler equations, an extension of the CM method has been proposed recently in Yin et al. [2021b].

# 7 Appendix

## 7.1 Hermite interpolation in 2D

Let's assume that we want to interpolate a function $u : \mathbb{R}^2 \to \mathbb{R}$ in a square $sw, se, nw, ne$ (c.f. figure 4) of side length $h > 0$, where the values of $u$ and its derivatives $\partial_x u, \partial_y u, \partial^2_{xy} u$ are known at the four vertices.

By rescaling the coordinates $x_1, x_2$ we have:

$$\tilde{x_1} = \frac{x_1 - x_1^{sw}}{h}, \quad \tilde{x_2} = \frac{x_2 - x_2^{sw}}{h} \tag{36}$$

Then we define the basis functions for Hermite polynomials :

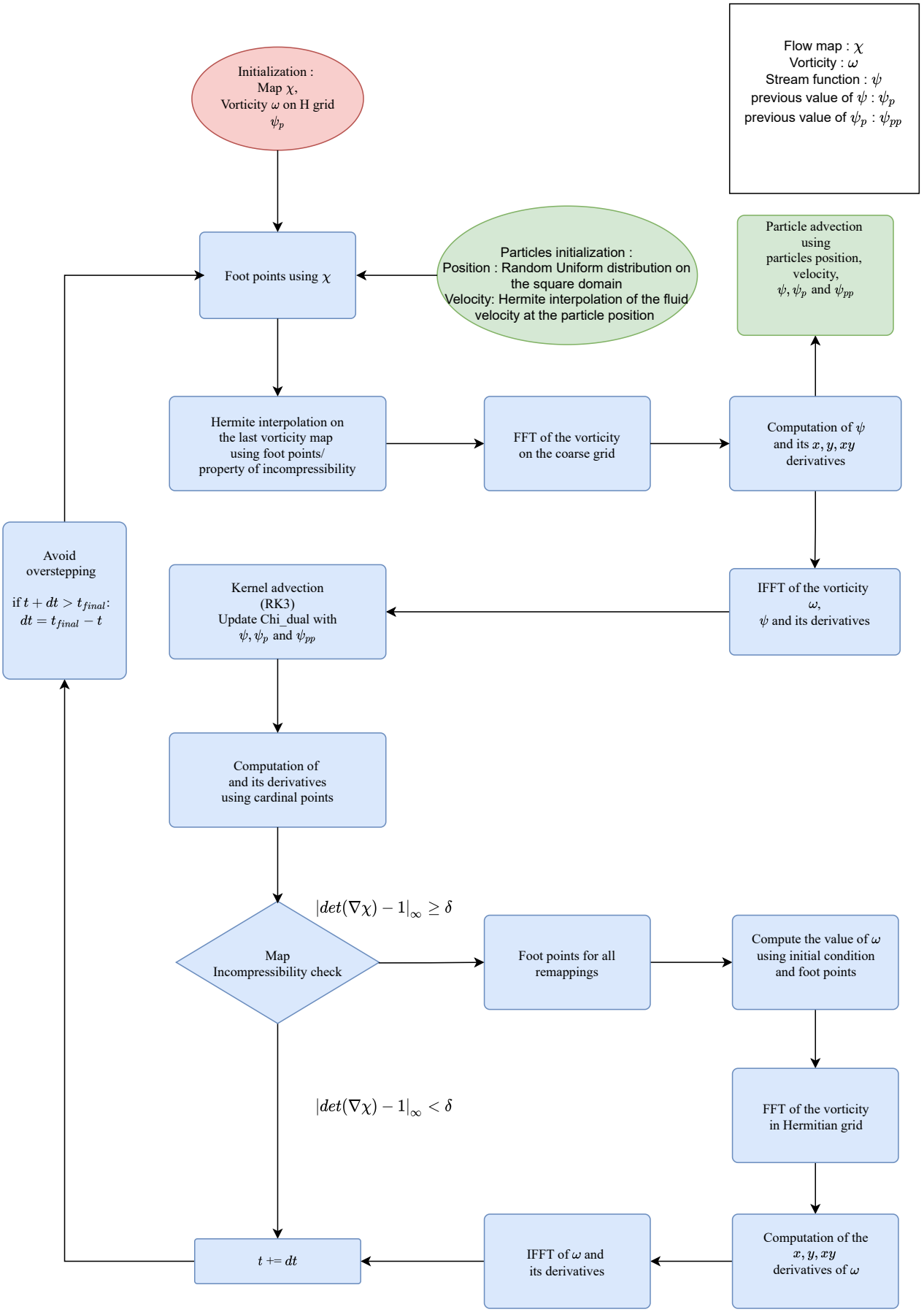
$$f(\tilde{x}) = 2\tilde{x}^3 - 3\tilde{x}^2 + 1, \quad g(\tilde{x}) = \tilde{x}^3 - 2\tilde{x}^2 + \tilde{x} \tag{37}$$

The value of $u$ at the point $(x_1, x_2)$ in the square $[x_1^{sw}, x_1^{se}] \times [x_2^{sw}, x_2^{se}]$ can be interpolated by the following formula [Kolomenskiy et al., 2016] :

$$\begin{aligned}
\tilde{u}(x_1, x_2) = \quad & u^{sw} f(\tilde{x_1}) f(\tilde{x_2}) + \quad u^{se} f(1-\tilde{x_1}) f(\tilde{x_2}) + \quad u^{nw} f(\tilde{x_1}) f(1-\tilde{x_2}) + \quad u^{ne} f(1-\tilde{x_1}) f(1-\tilde{x_2}) \\
& + h( \quad u^{sw}_{x_1} g(\tilde{x_1}) f(\tilde{x_2}) - \quad u^{se}_{x_1} g(1-\tilde{x_1}) f(\tilde{x_2}) + \quad u^{nw}_{x_1} g(\tilde{x_1}) f(1-\tilde{x_2}) - \quad u^{ne}_{x_1} g(1-\tilde{x_1}) f(1-\tilde{x_2}) \ ) \\
& + h( \quad u^{sw}_{x_2} f(\tilde{x_1}) g(\tilde{x_2}) + \quad u^{se}_{x_2} f(1-\tilde{x_1}) g(\tilde{x_2}) - \quad u^{nw}_{x_2} f(\tilde{x_1}) g(1-\tilde{x_2}) - \quad u^{ne}_{x_2} f(1-\tilde{x_1}) g(1-\tilde{x_2}) \ ) \\
& + h^2( \ u^{sw}_{x_1 x_2} g(\tilde{x_1}) g(\tilde{x_2}) - \ u^{se}_{x_1 x_2} g(1-\tilde{x_1}) g(\tilde{x_2}) - \ u^{nw}_{x_1 x_2} g(\tilde{x_1}) g(1-\tilde{x_2}) + \ u^{ne}_{x_1 x_2} g(1-\tilde{x_1}) g(1-\tilde{x_2}) \ )
\end{aligned}$$

The same can be done for the derivatives and cross derivative of $u$. This technique is used in the code and is explained in more details in the appendix of [Kolomenskiy et al., 2016].

## 7.2 Flowchart



## References


Claude W Bardos and Edriss S Titi. Mathematics and turbulence: where do we stand? *Journal of Turbulence*, 14(3):42–76, 2013.

Guido Boffetta and Robert E Ecke. Two-dimensional turbulence. *Annual review of fluid mechanics*, 44:427–451, 2012.

Marie Farge. Choice of representation modes and color scales for visualization in computational fluid dynamics. In *Science and Art Symposium 2000*, pages 91–110. Springer, 2000.

J. S. Ferenc and Z. Néda. On the size distribution of Poisson Voronoi cells. *Physica A: Statistical Mechanics and its Applications*, 385(2):518–526, 2007.

Uriel Frisch. *Turbulence: the legacy of AN Kolmogorov*. Cambridge University Press, 1995.

Dmitry Kolomenskiy, Jean-Christophe Nave, and Kai Schneider. Adaptive gradient-augmented level set method with multiresolution error estimation. *Journal of Scientific Computing*, 66(1):116–140, 2016.

Marcel Lesieur. *Turbulence in fluids: stochastic and numerical modelling*, volume 488. Nijhoff Boston, MA, 1987.

Martin R Maxey and James J Riley. Equation of motion for a small rigid sphere in a nonuniform flow. *The Physics of Fluids*, 26(4):883–889, 1983.

Alfons Michalke. The instability of free shear layers. *Progress in Aerospace Sciences*, 12:213–216, 1972.

Olga Podvigina, V Zheligovsky, and Uriel Frisch. The Cauchy–Lagrangian method for numerical analysis of Euler flow. *Journal of Computational Physics*, 306:320–342, 2016.

Witold Wolibner. Un theorème sur l'existence du mouvement plan d'un fluide parfait, homogène, incompressible, pendant un temps infiniment long. *Mathematische Zeitschrift*, 37(1): 698–726, 1933.

Badal Yadav. Characteristic mapping method for incompressible Euler equations. Master's thesis, McGill University, Canada, 2015.

Xi-Yuan Yin, Olivier Mercier, Badal Yadav, Kai Schneider, and Jean-Christophe Nave. A characteristic mapping method for the two-dimensional incompressible Euler equations. *Journal of Computational Physics*, 424:109781, 2021a.

Xi-Yuan Yin, Kai Schneider, and Jean-Christophe Nave. A characteristic mapping method for the three-dimensional incompressible Euler equations. *arXiv preprint arXiv:2107.03504*, 2021b.