Investigation of mixing and particle transport in 2D incompressible Euler flows using the characteristic mapping method

Julius Bergmann

Centre national de la recherche scientifique
Institut de Mathématiques de Marseille

Investigation of mixing and particle transport in 2D incompressible Euler flows using the characteristic mapping method

Master's thesis submitted in fulfilment of the requirements for the degree of
M. Sc. Physikalische Ingenieurwissenschaft

Author:
Julius Bergmann

Submission:
Berlin, April 20, 2022

Examiner:
Prof. Dr. rer. nat. Julius Reiss
Prof. Dr. Kai Schneider

Declaration of independence

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Je déclare et confirme que cette thèse est entièrement le résultat de mon propre travail original. Lorsque d'autres sources d'information ont été utilisées, elles ont été indiquées comme telles. Je déclare également que ce travail ou un travail similaire n'a pas été soumis ailleurs.

Berlin, April 20, 2022

Julius Bergmann

Abstract

Exponentially growing vorticity gradients in computations of turbulent flow governed by the two-dimensional incompressible Euler equations impose huge constraints for conventional numerical solution methods. The Semi-Lagrangian characteristic mapping method proposes a new approach to overcome this issue by evolving the flow map along characteristic curves. With the possibility to decompose the flow map into several sub-maps due to its semi-group structure and the gradient-augmented level set method to combine individual discrete sub-maps together, this method is able to capture the exponentially growing gradients. It achieves exponential resolution in linear time, enabling strong magnification of regions of interest until machine precision. This thesis deals with the extension of the work previously done. Strong focus is done in continuing the code development, initially written by Badal Yadav, to solve the characteristic mapping method efficiently on GPUs. The code is fully reworked and validated according to modern programming standards and recent mathematical advancements for the method. Afterwards, the implementation of fluid and inertial point particles is stated and validated. At last, applications of the code for further research interests are presented in the form of flow of passive scalar in shear layer flow and inertial particle behaviour in artificial homogeneous isotropic turbulence. The framework proves to be highly flexible and efficient in order to deal with further open topics regarding the incompressible Euler equation. The particle implementation enables research of more advanced immersed particle problems, such as the fractality of particle clusters in turbulent flow.

Kurzfassung

Exponentiell wachsende Gradienten der Wirbelstärke in Simulationen von turbulenten Strömungen bestimmt durch die zweidimensionalen inkompressiblen Eulergleichungen bewirken starke Einschränkungen für herkömmliche numerische Lösungsmethoden. Die Semi-Langrangesche Characteristic Mapping Methode (CMM) stellt einen neuen Ansatz dar um diese Limitierung zu überwinden indem es die Strömungskarte entlang charakteristischer Kurven entwickelt. Mit der Möglichkeit die Strömungskarten aufgrund der Halbgruppenstruktur in mehrere Teilkarten aufzuteilen und der Gradient-Augmented Level Set Methode um die einzelnen diskreten Teilkarten wieder zusammenzuführen ist die CMM fähig die exponentiell wachsenden Gradienten aufzulösen. Sie erreicht exponentielle Auflösung in linear Zeit, was eine Vergrößerung von wichtigen Bereichen bis zum Maschinenfehler ermöglicht. Diese Abschlussarbeit beschäftigt sich mit der Fortführung des Computercodes, ursprünglich entwickelt von Badal Yadav, um die Characteristic Mapping Methode effizient auf GPUs zu lösen. Der Code ist komplett überarbeitet und validiert übereinstimmend mit modernen Programmierungsstandards und aktueller mathematischen Fortschritten für die Methode. Anschließend wird die Implementierung von Interial- und Fluidpunktpartikeln gegeben und validiert. Zuallerletzt werden Anwendungen des Codes an weitern Forschungsinteressen präsentiert in der Form von Transport und Vermischung von passiven Skalaren in Scherschichtströmungen und dem Verhalten von inertiellen Partikeln in künstlicher homogener und isotroper Turbulenz. Das Rahmenwerk zeichnet sich durch seine hohe Flexibilität und Effizienz aus um für weitere offene Themenfelder der inkompressiblen Eulergleichungen genutzt zu werden. Die Partikelimplementierung er-möglicht die Untersuchung von komplexeren Problemen von eingebetten Partikeln von endlicher Grösse, wie der Fraktalität von Partikelgruppen in turbulenten Strömungen.

Résumé

La croissance exponentielle du gradient de vorticité dans les calculs de turbulence pour les équations d'Euler incompressibles bidimensionnelles impose d'énormes contraintes aux méthodes conventionnelles de résolution. La méthode d'application caractéristique semi-lagrangienne propose une nouvelle approche pour surmonter ce problème en faisant évoluer l'application de l'écoulement le long de courbes caractéristiques. Via la possibilité de décomposer l'application de l'écoulement en plusieurs sous-applications grâce à la structure de semi-groupe ainsi qu'à la méthode GALS pour composer ensemble les sous-applications discrètes, cette méthode est capable de prendre en compte la croissance exponentielle du gradient. Elle atteint une résolution exponentielle en temps linéaire, permettant un fort grossissement des régions d'intérêt jusqu'à la précision de la machine. Ce mémoire porte sur l'extension de travaux réalisés précédemment. Une attention particulière est portée à l'extension du code, initialement développé par Badal Yadav, pour résoudre la méthode d'application caractéristique sur les GPU. Le code est entièrement révisé et validé selon les standards de programmation actuels et les avancées mathématiques récentes de la méthode. De plus, des particules ponctuelles fluides et inertielles ont été implémentées et validées. Enfin, des applications du code à d'autres sujets de recherche sont présentées comme l'écoulement d'un scalaire passif dans des couches de cisaillement ainsi que le comportement de particules inertielles dans une turbulence homogène isotrope. Le cadre s'avère très flexible et efficace pour traiter d'autres sujets ouverts concernant l'équation d'Euler incompressible. L'implémentation des particules permet d'étudier des problèmes plus avancés, tels que la fractalité des amas de particules dans un écoulement turbulent.

Contents

Notation

All important notations and symbols can be found in this section. They will be stated in three-dimensions for completeness. In order to retrieve the two-dimensional equivalents, all terms containing the z-coordinate can be omitted and vectors become two-dimensional. Special caution has to be taken for the vorticity $\vec{\omega}$, which in 2D reduces to a pseudo-scalar $\omega := \omega_z$

ρ	Density of the fluid
p	Pressure of the fluid
θ	Passive scalar
Ψ	Stream function of the fluid
χ^+	Forward characteristic flow map
χ^-	Backwards characteristic flow map
$\vec{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$	Cartesian coordinate directions
$\vec{k} = \begin{pmatrix} k_x \\ k_y \\ k_z \end{pmatrix}$	Wavelengths in Fourier space
$\vec{u} = \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix}$	Velocity of the fluid flow
$\vec{\omega} = \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix}$	Vorticity of the fluid
$\nabla f = \begin{pmatrix} \partial_x f \\ \partial_y f \\ \partial_z f \end{pmatrix}$	Gradient of a scalar function f
$\nabla^\perp = \begin{pmatrix} -\partial_y \\ \partial_x \end{pmatrix}$	Perpendicular gradient in two dimensions
$\nabla \times \vec{f} = \begin{pmatrix} \partial_z f_y - \partial_y f_z \\ \partial_x f_z - \partial_z f_x \\ \partial_y f_x - \partial_x f_y \end{pmatrix}$	Curl of a vector function $\vec{f}$
$\nabla \cdot \vec{f} = \partial_x f_x + \partial_y f_y + \partial_z f_z$	Divergence of a vector function $\vec{f}$
$\Delta f = \nabla \cdot \nabla f = \partial_{xx} f + \partial_{yy} f + \partial_{zz} f$	Laplacian of a scalar function f

1 Introduction

Fluid flow is an important and fascinating phenomenon encountered in nature all around us. One feature of fluid flow, being turbulence, is still to this day a widely researched topic with many important applications. Its difficulty lies in the broad range of scales, where kinetic energy of the fluid flow forms large scale coherent structures emerging from bodies, which is continuously broken down into smaller scales and at last dissipated to heat due to viscous friction at very fine scales. This makes simulations challenging, as often a very high resolution in space and time is needed to represent all relevant physical phenomena. This becomes even more challenging when dealing with inviscid flow, where no dissipation of energy takes place and it is broken down into infinitesimally small structures. As current simulation techniques are hardly able to appropriately represent those scales, novel methods are needed. The characteristic mapping method used in this work makes up for a good approach with its possibility to resolve exponential resolution in space within linear time.

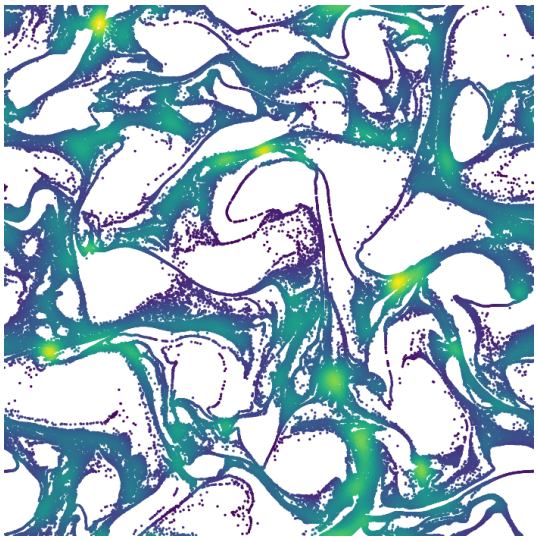

Figure 1: Clustering of light point particles embedded into turbulent flow computed with the characteristic mapping method

In 2009, Nave et al. developed the gradient-augmented level set (GALS) method in [14]. This method presented an optimal way to transport a level set function together with its derivatives, where the level set method itself can represent a surface as the zero contour of the level set function. With the presented method, approximations of curvature information were preserved with tracking local structures smaller than the grid size and later using Hermite interpolation for reconstruction. Being initially used to represent surfaces, this was quickly extended to track transport along characteristics. In [7], it was compared to other common methods to solve linear advection along characteristics, where it proved to be highly efficient. From this, Mercier et al. constructed the characteristic mapping method (CMM) for the linear advection of arbitrary sets [13]. It allowed transport of sets due to a velocity field to be computed to high accuracy with low computational cost and was achieved by computation of long-term deformation maps. The characteristic maps, which basically describe the movement of individual positions within the sets, form a sub-group structure. Together with the Hermite interpolation from the GALS method, each long-term deformation can be separated into small individual sub-maps, where the

positions can be interpolated in order to link the maps together. This allows the representation of the solution to arbitrarily fine sub-grid resolution.
At a meeting between Prof. Nave and Prof. Schneider, the idea sparked to extend this for the 2D incompressible Euler equations. As there the pseudo-scalar vorticity is merely transported, the characteristic mapping method can be used in order to solve the vorticity and velocity over time. A first framework to solve this, from which the presented framework origins from, was developed by Badal Yadav in the scope of his master thesis at the McGill university in Canada in 2015 [19]. This featured the mathematical framework being implemented on C++ and Nvidia Cuda to be solved efficiently on graphics card. First result showed stunning results, especially with the possibility to magnify areas until machine precision. The mathematical formulation was further extend by Xi-Yuan Yin during his doctorate. He first extensively formulated and validated the method to be used as a Semi-Lagrangian approach to solve the 2D incompressible Euler equations in [21]. It was later extended to 3D by the use of Lie algebra in [22]. There, the vortex stretching term is included into the characteristic maps to practically couple the vorticity in all three dimensions. He also conducted tests in order to investigate how diffusion effects can be computed by the means of the characteristic mapping method in [20]. Currently, the CM-method for the 2D incompressible Euler equations is adapted for applications and further refinement. This includes the research in an on-going project funded by the Agence Nationale de la Recherche (ANR) in France. There, the method should be further enhanced by the usage of multi-resolution techniques and further applications in 3D. Included in the project was also the master thesis by Nicolas Saber in [15], who implemented advected point particles into the 2D code initially developed by Badal Yadav. His master project preceded the work presented in this thesis, which is also funded by the ANR project.

The end of the thesis of Nicolas Saber also marked the beginning of this work. The main work was centred around the framework and code as the main tool to solve and include further applications. However, it still lacked several features developed by Xi-Yuan Yin. In comparison to his work the code was not yet validated and even missed key features as in example the higher order time-stepping methods. In addition, the code initially from Badal Yadav was still in prototype form and therefore barely suitable to be used for special applications. At last, the particle implementation by Nicolas Saber was included, but a thorough validation was yet to be done.
This all together formed the first tasks for this thesis. The code and framework had to be cleaned and reworked in regards to recent advancements and more accessible coding standards. In addition, a thorough validation was needed in order to verify the correctness of both the implementation of the CM-method itself and the point particle implementation.
After this has been concluded, several aspects were considered as possible physical extensions to the framework. From those the inclusion of passive scalar quantities as

an additional feature to the framework and further work with inertial point particles were chosen. With that, several initial conditions also have been reworked in order to better represent physically significant conditions.

This thesis is divided into several chapter, each explaining different aspects that have been used, studied or investigated. It starts with an explanation of the mathematical background in chapter 2. The 2D incompressible Euler equations are derived in vorticity form and the characteristic mapping method to solve it is presented. Chapter 3 gives an introduction to the Cuda framework, which is used to solve the CM-method on graphics cards. Additionally, the work on the code which was done will be presented. Thereafter, starting from chapter 4, my work on and with the framework will be further presented. In the chapter itself, several mathematical improvements will be explained. The code is validated in space and time and important hyper-parameters will be discussed. The implementation of finite sized fluid and inertial particles will be introduced and validated in chapter 5. The following two chapters deal with specific implementations, each focusing on a new aspect while dealing with a specific initial condition. In chapter 6, the computation of passive scalars will be introduced. Those will be included in the simulation of a shear layer flow. The zoom property will be used to more precisely investigate the mixing behaviour of shear flow. The next chapter 7 deals with fluid and inertial particles embedded in 2D homogeneous isotropic turbulence. The inertial particles are further classified and their Stokes number will be validated. Afterwards, the fractality of a ring-structure will be investigated under turbulent flow. At last in chapter 8, the results of this work will be summarised. Limitations will be given and further research ideas will be listed.

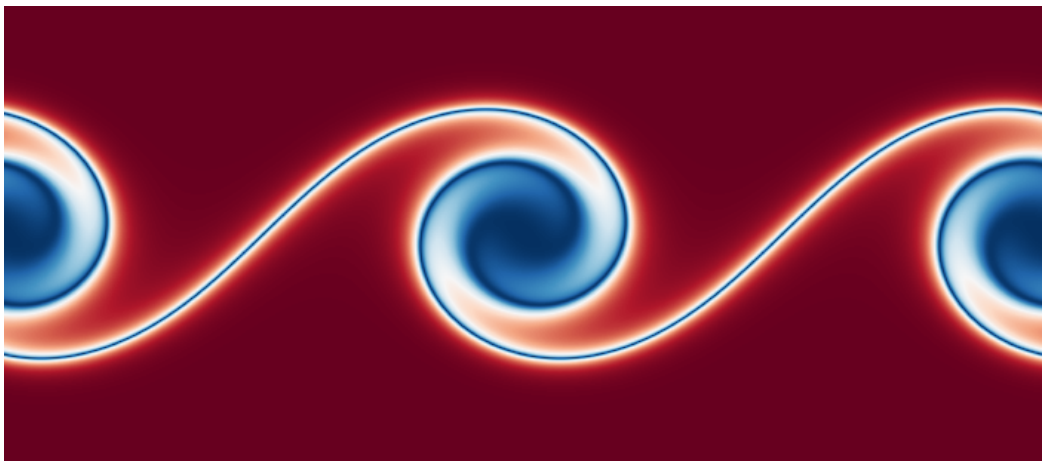

Figure 2: Kelvin-Helmholtz instabilities developing in the vorticity from shear layer flow computed with the characteristic mapping method. Regions of high vorticity are depicted in blue.

2 Mathematical framework

Not only do many different ways to formulate the behaviour of fluid flow exist, but there are different ways to solve each individual formulation. Most features of real fluids can be described by the famous Navier-Stokes-equation, which were developed in the beginning of the 19th century and are given in Equation (2.1) and (2.2) in convective form. ([10], p.20)

$$\frac{d\rho}{dt} = \partial_t \rho + \nabla \cdot \vec{u} = 0 \tag{2.1}$$

$$\rho \frac{d\vec{u}}{dt} = \rho \partial_t \vec{u} + \vec{u} \cdot \nabla \vec{u} = -\nabla p + \nabla \cdot \boldsymbol{\tau} \tag{2.2}$$

These equations describe the conservation of mass and momentum in three dimensions on a flow domain $\Omega \in \mathbb{R}^3$ with increasing time $t \in \mathbb{R}^+$ and further respective boundary and initial conditions. With the total derivative, the momentum equation greatly resembles Newton's second law of motion, defining the acceleration of the fluid as the sum of pressure and stress forces acting on it.
The two biggest factors giving rise to different fluid flow properties is the compressibility of the density ρ and the viscosity due to the stress tensor $\boldsymbol{\tau}$ acting on the fluid. The first mainly differentiates gaseous from liquid flows and also largely changes how structural information propagates inside a fluid. The main parameter for this is the Mach number, describing the ratio of fluid velocity to the velocity of propagation of information within the fluid. The second can give rise to highly complex phenomena like turbulence and, together with the governing Reynolds number describing its strength, is a key parameter in modern engineering fluid flow investigations.
In the following sections, the incompressible Euler equation will be derived as a simplification of the Navier-Stokes equations. Later, the equations will be reformulated in vorticity form and the characteristic mapping method as a solution approach will be presented in detail.
Afterwards, further elaborations on the numerical discretisation in space and time will also be given for the derived set of equations.

2.1 Derivation of the 2D Incompressible Euler equations in vorticity form

The given framework computes solutions to the incompressible Euler equations. These set of equations describe the conservation property of perfect, incompressible fluids and consist of the conservation equations for the mass and momentum, similar to the Navier-Stokes equations.
In comparison to the full set of Navier-Stokes equations both friction and compressibility are neglected. An incompressible fluid has a constant density over time $\rho = const.$, so equation (2.1) breaks down to a divergence free condition for the velocity. This simplification assumes the Mach number to be close or equal to zero and information like a change in pressure or velocity travels infinitely fast through

the domain. By assuming the density to be equal to 1, it can be eliminated from all equations. A friction-less fluid has no shear stress acting on it, which eliminates the stress term from the momentum equation (2.2). This would coincide to a Reynolds number limit of infinity. No energy is lost due to friction over time and the flow becomes reversible. Together, the incompressible Euler equations read the following:

$$\nabla \cdot \vec{u} = 0 \tag{2.3}$$

$$\frac{d\vec{u}}{dt} = \partial_t \vec{u} + \vec{u} \cdot \nabla \vec{u} = -\nabla p \tag{2.4}$$

Here, the first equation describes the conservation of mass via a divergence free condition for the velocity. The second equation is the conservation of momentum. With the exemption of the friction term, this equation becomes similar to a nonlinear transport equation, where the pressure gradient forms a numerical source term for the motion of the fluid. In fact, it represents a Lagrangian multiplier imposing the incompressibility condition.
For the used framework, the equations are defined on a 3-flat Torus $\mathbb{T}^3$ with periodic boundary conditions in order to easily define all quantities as Fourier series. The initial state is only given by the initial velocity

$$\vec{u}(t=0) = \vec{u}_0 \tag{2.5}$$

As no time derivative for the pressure is present, no initial condition has to be imposed for it.
The equations (2.3) and (2.4) form an idealised case, as real fluids are neither perfectly incompressible nor completely friction-less. However, many present phenomena in real fluids can be approximated by using this formulation and especially behaviour of real turbulence can be well approximated. All together the characteristics of these equations are very unique, as only the velocity is transported in respect to the pressure gradients.

As turbulent flow consists of many vortical structures, it is beneficial to rewrite the transport equation for the velocity (2.4) to the transport of vortical motion. The derivation will be done similar to [21] and [15]. The vorticity is defined as the rotation of a fluid at any given point. It is connected to the velocity by $\vec{\omega} = \nabla \times \vec{u}$. Applying the curl operator to the momentum equation, one can retrieve the transport equation for the vorticity together with its initial condition:

$$\frac{d\vec{\omega}}{dt} = \partial_t \vec{\omega} + (\vec{u} \cdot \nabla)\vec{\omega} = (\vec{\omega} \cdot \nabla)\vec{u} \tag{2.6}$$

$$\vec{\omega}\,(t=0) = \vec{\omega}_0 \tag{2.7}$$

The curl of a gradient is always zero, the pressure term therefore vanishes after application of the curl operator. The vorticity is solely transported, with the vortex stretching term $(\vec{\omega} \cdot \nabla)\vec{u}$ describing the flow between the coordinate directions.

While the vorticity for each coordinate direction is not conserved on the computational domain, it is in respect to all dimensions.

For further derivation the flow is assumed to be two-dimensional in x- and y-direction. In planar flow, the vorticity simplifies greatly. Due to no velocity being present in z-direction, the vorticity vector reduces to

$$\vec{\omega} = (0, 0, \omega_z)^T \tag{2.8}$$

The vorticity therefore reduces to a pseudo-scalar $\omega := \omega_z = \partial_y u_x - \partial_x u_y$. By definition it is perpendicular to the planar flow. Due to that, the vortex stretching term $\vec{\omega} \cdot \nabla \vec{u}$ vanishes and the transport of vorticity becomes

$$\frac{d\omega}{dt} = \partial_t \omega + \vec{u} \cdot \nabla \omega = 0 \tag{2.9}$$

In order to later solve the equations, a formulation for the velocity from the vorticity is beneficial. This is achieved by connecting both quantities via the stream function Ψ. The existence of the stream function arises from the divergence free condition for the velocity in two-dimensional flow.

$$\vec{u} = \nabla^\perp \Psi \tag{2.10}$$

$$\omega = \Delta \Psi \tag{2.11}$$

The operator $\nabla^\perp = (-\partial_y, \partial_x)^T$ is the perpendicular gradient in two dimensions and the condition for the scalar vorticity comes from taking the 2D-curl of the prior equation (2.10). Both equations (2.10) and (2.11) can be combined to form the so-called Biot-Savart law:

$$\vec{u} = \nabla^\perp \Delta^{-1} \omega \tag{2.12}$$

Here Δ^{-1} is the inverse Laplacian, which is well defined for smooth functions and can be properly constructed utilising Green's functions.
All-together, the total set of equations read as:

$$\begin{aligned} \frac{d\omega}{dt} = \partial_t \omega + \vec{u} \cdot \nabla \omega = 0 \\ \nabla \cdot \vec{u} = 0 \\ \vec{u} = \nabla^\perp \Delta^{-1} \omega \\ \omega\,(t = 0) = \omega_0 \end{aligned} \tag{2.13}$$

For the boundary conditions, the domain is assumed to be on the 2-flat Torus $\mathbb{T}^2$ with periodic boundary conditions. Due to that, a Fourier series representation of all quantities can be defined which are used to solve the Biot-Savart law in Fourier space which is shown and explained later in Section (2.3.5).

2.1.1 Behaviour of global quantities

Important quantities to study for physical flow are global quantities being defined over the whole domain. For planar flow, the energy, enstrophy and palinstrophy are of interest. These formulate the change in total kinetic energy, vorticity and vorticity gradients over the whole domain and are defined as

$$E(t) = \frac{1}{2}\int_{\mathbb{T}^2} |\vec{u}(\vec{x},t)|^2 d\vec{x} \tag{2.14}$$

$$Z(t) = \frac{1}{2}\int_{\mathbb{T}^2} |\omega(\vec{x},t)|^2 d\vec{x} \tag{2.15}$$

$$P(t) = \frac{1}{2}\int_{\mathbb{T}^2} |\nabla\omega(\vec{x},t)|^2 d\vec{x} \tag{2.16}$$

As the computational domain with periodic boundaries is formulated as a closed system, mass and energy have to be conserved in respect to the whole domain. The first leads to a preservation of volume, as the density is set to be constant. The latter is a useful condition in order to test numerical schemes and is usually hard to achieve with artificial diffusion present in many applications, leading to a loss in global energy over time. Derived from the transport of vorticity for two-dimensional flow in equation (2.9), the scalar vorticity is also conserved globally. Both the global conservation of energy and enstrophy can be mathematically taken from the fulfilled transport equations without source terms:

$$\frac{dE(t)}{dt} = 0 \qquad \frac{dZ(t)}{dt} = 0 \tag{2.17}$$

$$d_t P(t) = -2\int_{\mathbb{T}^2} \theta\cdot\nabla\theta\cdot u d\vec{x} \quad \text{where} \quad \theta = \nabla^{\perp}\omega \tag{2.18}$$

However, the palinstrophy is not globally conserved. In fact, for solutions of the Euler equations it increases exponentially over time. This is due to the fact, that the gradient of vorticity increases exponentially over time, with the vorticity continuously breaking down into finer and finer scales. This is also called the enstrophy cascade, as explained in ([10], p. 183).
As the computational domain is the 2-flat torus $\mathbb{T}^2$, all quantities can be represented as a super-composition of Fourier modes $\vec{k}$ in all cardinal directions. Due to this representation, the energy, enstrophy and palinstrophy can also be represented the same way as the integration over all wave-numbers.
In order to quantify and investigate the quantities of isotropic turbulence in Fourier space, the isotropic spectrum is used. This combines the wave-numbers in both directions by $k = \sqrt{k_x^2 + k_y^2}$. Later, one-dimensional spectra of the energy, energy and palinstrophy can be computed in respect to the norm of the wave numbers k. This will later be a useful representation in order to show-case the emerging energy and enstrophy cascades, as for isotropic turbulent flow the intensity is the same in all directions.

2.2 Characteristic mapping method

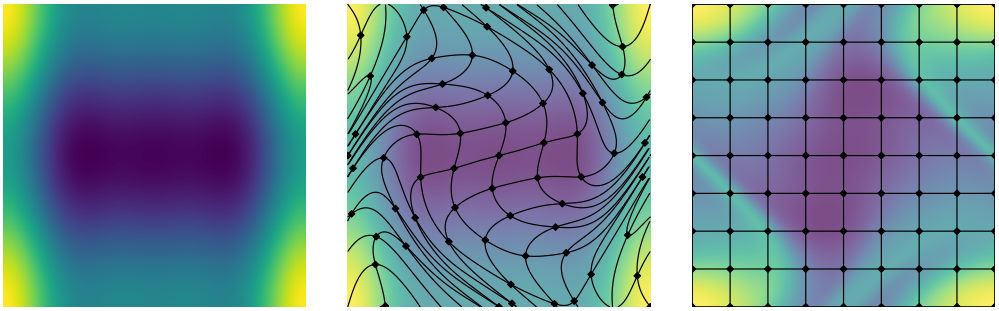

Figure 3: Pullback of the vorticity from the initial condition (left), overlay of the deformed backwards flow map (middle) and reformed vorticity with uniform initial map (right).

All together the equations (2.13) form a closed set in an Eulerian frame. One would be able to discretise the vorticity on a fixed mesh and advect it over time by evaluating the velocity.
However, as the flow is invertible and incompressible, it can also be discretised in a more specialised way by computing the movement of the fluid particles.
One way to do that is by using the characteristic mapping method, which was first introduced by Mercier et al. in [13] for the linear advection of arbitrary sets. The movement of individual flow particles is tracked back to the initial condition where they originate from. This can be done since the flow merely transports fluid particles while preserving the volume, leading to no actual distortion or diffusion on the domain. Further refined by Yin et al. in [21], this was adapted to work with the 2D incompressible Euler equation. This section aims to derive the characteristic mapping method for the 2D incompressible Euler equations similar to [21].

The momentum equation for the vorticity in the equations (2.13) consists merely of the advection of the vorticity by the velocity. This forms a strongly non-linear form of a transport equation, however, normal properties of transported sets still apply. As no source term is present, one can define the solution of the vorticity at any time by transporting along characteristic curves $\vec{\gamma}$:

$$\omega(\vec{\gamma}(t), t) = \omega_0(\vec{\gamma}(0)) \tag{2.19}$$

This essentially describes a Lagrangian form of the solution, where it can be interpreted as fluid particles being transported along a characteristic curve, starting from initial positions $\vec{\gamma}_0$. The characteristic itself can be defined from the velocity as:

$$\frac{d\vec{\gamma}(t)}{dt} = \vec{u}(\vec{\gamma}(t), t) \tag{2.20}$$

$$\vec{\gamma}(0) = \vec{\gamma}_0 \tag{2.21}$$

To solve the vorticity at an arbitrary time t, it is therefore enough to only integrate the characteristic curves along the velocity. As a result, the solution at any time is only a movement from the initial condition along the velocity of a fluid particle. This way of solving the transport equation is called method of characteristics and is commonly applied to solve transport equations. A huge benefit of this formalism is, that many different integration techniques are globally stable for Lagrangian formulations [1].
All characteristic curves $\vec{\gamma}(t)$ together form characteristic mappings defined on the whole domain. Due to the invertibility of the flow, these can be defined forward and backwards in time, so that

$$\chi^+(\vec{\gamma}_0, t) = \vec{\gamma}(t) \tag{2.22}$$
$$\chi^-(\vec{\gamma}(t), t) = \vec{\gamma}_0 \tag{2.23}$$

In equation (2.22), χ^+ is the forward flow map which describes the mapping from an initial position $\vec{\gamma}_0$ at time 0 to an advected position $\vec{\gamma}(t)$ at time t, practically combining all characteristic curves. Accordingly, χ^- as the backwards flow map describes the mapping from an advected position at time t back to its initial position, practically forming a pullback to the initial state. With this, one is able to compute from which position on the initial condition a fluid particle is advected to the given position. One can easily see, that both maps applied in sequence reduce to the identity, as

$$\chi^-(\chi^+(\vec{\gamma}_0, t), t) = \vec{\gamma}_0 \tag{2.24}$$

As the velocity has to satisfy the divergence-free condition and are assumed to be smooth, no two characteristic curves will collide to any given time. Due to that, the two flow maps form diffeomorphisms, uniquely mapping one point of the domain to another and therefore the characteristic maps satisfy a transport condition on their one:

$$\partial_t \chi + \vec{u} \cdot \nabla \chi = 0 \tag{2.25}$$
$$\chi(\vec{x}, 0) = \vec{x} \tag{2.26}$$

Here the initial condition of the maps is the identity, mapping any point on the domain to itself. The solution of the vorticity at any position at time t can now conveniently be described by utilising the backwards flow map

$$\omega(\vec{x}, t) = \omega_0(\chi^-(\vec{x}, t)) \tag{2.27}$$

An illustration for this can be found in figure 3. The image on the right represents a uniform grid as the initial condition for the backwards flow map at time t. Using the backward flow map, each position can be traced back to where they originally come

from the initial condition, which is shown in the center image. By doing this for the whole plane, the initial condition shown in the left image is essentially distorted, shown again in the right image.
While before in the equations (2.13) the vorticity itself was transported, forming an Eulerian and static way of solving the equations, now we only advect the characteristic map and get the vorticity by forming a pullback to the initial condition. Using the Biot-Savart-law, the velocity can be computed from the vorticity, which is again used to advect the characteristic map. The set of equations to be solved changes to

$$\begin{aligned} \omega(\vec{x},t) &= \omega_0(\chi^-(\vec{x},t)) \\ \vec{u} &= \nabla^\perp \Delta^{-1}\omega \\ \partial_t \chi^- + \vec{u}\cdot\nabla\chi^- &= 0 \end{aligned} \tag{2.28}$$

being called the advection-vorticity coupling. As the incompressibility condition is not directly included in this set of equations, a strong condition has to be additionally imposed in order to retain it. For the characteristics maps, this is given by

$$\det(\nabla\chi) = 1 \tag{2.29}$$

This implies a volume preserving property for the characteristic mappings due to isometry, forming the pendant to the mass preserving property of the divergence-free condition in the Eulerian frame. It will be further referenced as the incompressibility condition for the flow map.
While in theory only the forward map could be utilised to track the movement of fluid particles from an initial time, it imposes some difficulties. To later compute the velocity with the help of Fourier series representations of the vorticity, it has to be given on a uniform grid at the current time-step. This can be achieved by using the backwards map, as we can easily define the grids at time t on which we want to pull back the initial condition. Together with using an analytical initial condition, this also enables to impose no loss of information from sampling from the initial conditions, even with very high distortion present. However, working with the backwards flow map also makes definitions more confusing, as it is always defined to map backwards in time. Special caution has to be taken when using integrational time-stepping methods, to take this into account.

2.3 Discretisation

In order to numerically solve the equations (2.28), they have to be discretised in space and time. This section gives further explanation on these specific parts. Firstly, the transport equation of the backwards flow map will be discretised and the used numerical methods will be presented, mainly the Hermite interpolation and gradient-augmented level set method (GALS). Afterwards, the remapping structure is explained in order to highlight the benefits of the chosen method. This also further explains how the vorticity is numerically reconstructed using the computed maps. A brief explanation of the computation of the Biot-Savart law is given and at last, a summary of the whole framework will be presented.

2.3.1 Discretisation of the backwards flow map in time

A characteristic flow map does not have to point from time $t = 0$ by definition, but could rather map a fluid particle from any arbitrary time instant to another, whereas the time direction is not important due to the invertibility of the flow. This property can be used, to basically divide a mapping from one time instant to another into smaller sub-intervals containing several maps applied in sequence and is called the semi-group structure. It is an important property of the flow maps.

$$\chi_{[0,\tau]} = \chi_{[\tau_2,\tau]} \circ \chi_{[\tau_1,\tau_2]} \circ \chi_{[0,\tau_1]} \tag{2.30}$$

The subscript denotes the interval boundaries, to or from which the flow map points to. The previously used notations for the backwards and forward map become $\chi^+(\vec{x},t) = \chi_{[0,t]}$ and $\chi^-(\vec{x},t) = \chi_{[t,0]}$. Using this expression, the full flow maps can be numerically adapted onto discrete time instants. The maps at further times $t+\Delta t$ with small time-steps Δt, can be computed from the flow maps at time t using the sub-composition:

$$\chi^+(\vec{x},t+\Delta t) = \chi_{[t,t+\Delta t]} \circ \chi^+(\vec{x},t) \tag{2.31}$$

$$\chi^-(\vec{x},t+\Delta t) = \chi^-(\vec{x},t) \circ \chi_{[t+\Delta t,t]} \tag{2.32}$$

Therefore, instead of evolving the flow map through the transport equation (2.25), it is always advanced through the semi-group structure. Also, already computed flow maps are used in order to then trace back to the initial position, as showcased in figure 4. This is the core of the characteristic mapping method and the reason to many of its beneficial features.

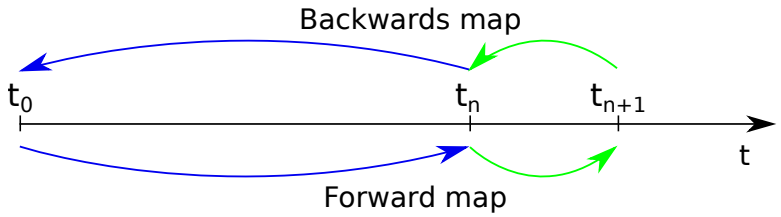

Figure 4: Computation of total map by application of previously computed map (blue) and integration to or from next timestep (green).

The flow maps for the discrete interval $[t+\Delta t, t]$ can be computed by individually advecting the fluid particles on each discrete position along the characteristic curves. The computational domain is discretised onto a finite, uniform grid, where each position is now advanced by solving the ordinary differential equation (2.20). This can be done using available numerical integration methods. In the presented framework Runge-Kutta methods were used, which will be further explained and discussed in section (4.1.2).

2.3.2 Hermite interpolation as integral bridge between Eulerian and Lagrangian frame

When discretising the flow map as explained in the last section, the backwards map is advanced from a uniform grid at time t_{n+1} to time t_n. This will lead to it being present at arbitrary positions, as also depicted in figure 5. However, at that time all variables are just available on another uniform grid, being previously computed. Important here are the velocity in order to actually advance the backwards map for the finite time and the previous backwards flow map in order to trace back to the initial position. This explains a common problem of semi-Lagrangian approaches, where some parts are defined on fixed grids, while others move alongside the flow. In order to combine those quantities, an interpolation method is needed. this will essentially build a bridge between the Lagrangian and Eulerian frame and allow the interpolation of needed quantities at arbitrary positions.

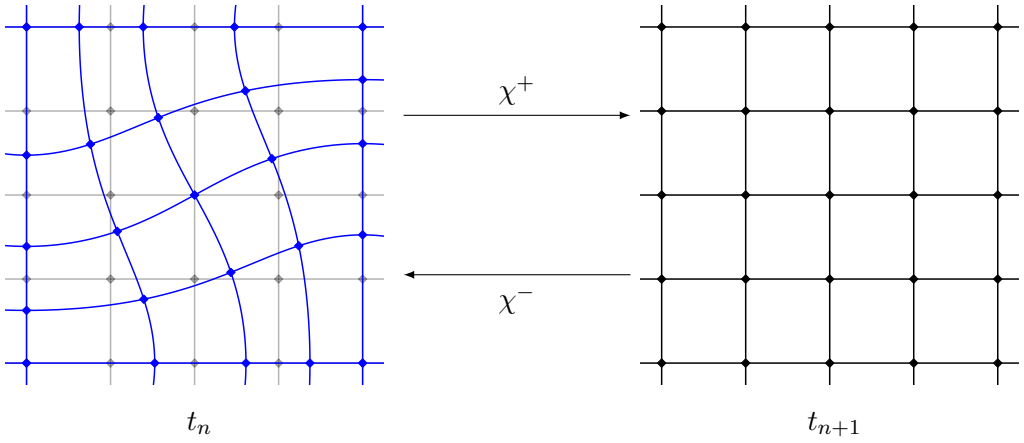

Figure 5: Uniform grid at time t_{n+1} (right) leads to arbitrary position at time t_n (left)

For the interpolation, a Hermite bicubic interpolation as done in [21] was chosen. As the procedure of this has been already presented in detail in the preceding literature, only a short overview will be given here together with the key features. The computational domain is spatially discretised on a uniform rectangular meshgrid with constant grid size $N_x, N_y \in \mathbb{N}$. Values of each quantities together with its first order normal derivatives in x- and y-direction and the first order mixed derivative are needed, in order to interpolate at an arbitrary position between the grid-points using the Hermite basis functions on the four neighbouring grid-points. Due to the property of the Hermite basis functions, each interpolated quantity is bicubic in each cell and everywhere differentiable with continuous mixed derivative $\partial x \partial y$. In addition, it has continuous normal derivatives and C^∞ tangential derivatives on cell boundaries.
The interpolation itself is of fourth order accuracy. Additionally, the continuous

first derivatives can be computed of third order accuracy. The Hermite interpolants are especially useful due to their compact form to achieve high-order estimates, as only the four adjacent grid-points are needed, making them well suited for fast but accurate implementations. Other interpolation methods of similar orders using multi-point approaches were found to be less optimal in terms of memory used over achieved accuracy.
In the implemented framework, the mappings, vorticity and stream function are used for Hermite interpolation. In order to compute the velocity for the advection of the flow maps, it is sampled from the stream function being of third order accuracy in space. All of those variables will be represented in their respective Hermite form, where the values are stacked together with their first order normal and cross derivatives.

2.3.3 Advection of derivatives of the flow map using the gradient-augmented level set method

While the advection of the map itself is straight forward using numerical integration techniques, one has to take into account the computation of the derivatives too. This is needed in order to represent the characteristic maps in Hermite form to be used for the Hermite interpolation and will be done by utilising the gradient-augmented level set (GALS) method, first presented in [14]. In this, a local stencil of sub-grid size around the uniform grid-points is chosen and advected. Afterwards, by utilising finite difference schemes, the map position and the derivatives are computed.

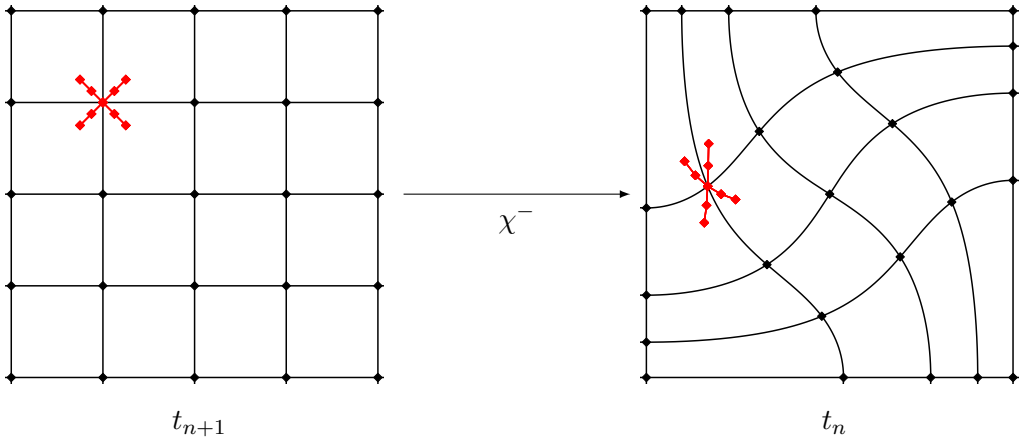

Figure 6: Local uniform stencil (left) is distorted after advecting backwards map to previous time instant (right).

In order to efficiently solve for the first order normal and mixed derivatives, a cross-shaped stencil proofs to be the most efficient in terms of needed points to advect, which is the largest factor to define the computational cost. The stencil can be increased in size to compute the quantities of higher order. A depiction of the

stencil can be seen in figure 6. Each point is located $[\pm i \cdot \epsilon_m, \pm i \cdot \epsilon_m]^T$ away from the grid point, where ϵ_m can be freely chosen. The factor $i \in \mathbb{Z}$ notes the stencil depth and ranges from one to the stencil size. Choosing a stencil size of n, $4n$ points have to be computed. With that, the quantity and all derivatives can be computed of order $2n$, as can easily be proven by insertion into the Taylor expansion. In order to not reduce the spatial accuracy, a size of $n \geq 2$ should be chosen to compute the map advection in at least fourth order, being consistent with the fourth order of the cubic Hermite interpolation being used.
The computation of the flow map values in Hermite form is given by

$$\begin{aligned}
\chi &= \sum_{i=1}^{n} \left[c_i \cdot \left(\chi_i^{\mathrm{NE}} + \chi_i^{\mathrm{SE}} + \chi_i^{\mathrm{NW}} + \chi_i^{\mathrm{SW}} \right) \right] \\
\partial_x \chi &= \sum_{i=1}^{n} \left[\frac{c_i}{\epsilon_m i} \cdot \left(\chi_i^{\mathrm{NE}} + \chi_i^{\mathrm{SE}} - \chi_i^{\mathrm{NW}} - \chi_i^{\mathrm{SW}} \right) \right] \\
\partial_y \chi &= \sum_{i=1}^{n} \left[\frac{c_i}{\epsilon_m i} \cdot \left(\chi_i^{\mathrm{NE}} - \chi_i^{\mathrm{SE}} - \chi_i^{\mathrm{NW}} + \chi_i^{\mathrm{SW}} \right) \right] \\
\partial x \partial y \chi &= \sum_{i=1}^{n} \left[\frac{c_i}{\epsilon_m^2 i^2} \cdot \left(\chi_i^{\mathrm{NE}} - \chi_i^{\mathrm{SE}} + \chi_i^{\mathrm{NW}} - \chi_i^{\mathrm{SW}} \right) \right]
\end{aligned}$$

Here, χ_i^{NE}, χ_i^{SE}, χ_i^{NW}, χ_i^{SW} represent the stencil points in north-east, south-east, north-west and south-west direction with their respective stencil depth. The factors c_i are the coefficients for the finite difference scheme. For $n = 1$, the only coefficient is $c_0 = \frac{1}{4}$, resulting in a 2D version of the classical second order central averaging scheme and central differences scheme for the normal derivatives.
Further analysis on improvements made on the actual map update will be given in section (4.2).

2.3.4 Sub-map structure and computation of vorticity

The use of the semi-group structure can not only be used to numerically discretise the grid, but also to further structure the backwards flow map. With the enstrophy cascade building up finer and finer scales for the vorticity for solutions to the Euler equations ([10], p. 183), the vorticity gradients $\nabla\omega$ increase exponentially too. The flow map, being linked to the vorticity, also experiences an exponentially increasing gradient $\nabla\chi$. This leads to strong deformation in specific parts of the computational domain. Using the composition structure of the map would eventually result in resets of the grid structure, leading to reduced error growth. This effect can be effectively repeated in order to adaptively adjust the numerical resolution of the flow map. The map acting on the total interval $[0, \tau]$ is divided into arbitrary many sub-intervals acting only on a relatively small time interval.
Each of these individual sub-maps can be computed and stored separately using the system of equations (2.28).

In order to bound the growth of error in each sub-map, a condition for remapping has to be defined. From the mathematical definitions, three numerical error quantities can be defined:

$$\delta_{inc,b}(\vec{x},t) = \det(\nabla\chi^{-}(\vec{x},t)) - 1 \tag{2.33}$$

$$\delta_{inc,f}(\vec{x},t) = \det(\nabla\chi^{+}(\vec{x},t)) - 1 \tag{2.34}$$

$$\delta_{inv}(\vec{x},t) = \chi^{-}(\chi^{+}((\vec{x},t)),t) - \vec{x} \tag{2.35}$$

The first two quantities describe the numerical incompressibility error from deviating from the incompressibility condition (2.29), while the last is the invertibility error from the invertibility condition (2.24). The first two give a measure of accumulation of error in each individual map and the invertibility error gives a measure for the connectivity of the two maps.
As the velocity is computed each step utilising only the backwards map, the error $\delta_{inc,b}$ was determined to be the most important for a remapping condition.
When the maximum incompressibility error of the current sub-map exceeds a specific thresh-hold, it's values are stored and a new map with the initial condition (2.26) will be used for further evaluations. In order to reduce computations, the initial condition from which the vorticity will be pulled back every step does not come directly from the condition at time $t = 0$. Rather, each latest sub-map is used to evolve the vorticity ω^* on a discrete fine grid. For each step, the current vorticity will be sampled from this quantity, which is represented in Hermite form to allow interpolation. With τ as the initial time for a given sub-map, the vorticity at the current time $t \geq \tau$ is computed by

$$\omega^*(\vec{x},\tau) = \omega_0(\chi^{-}(\vec{x},\tau)) \tag{2.36}$$

$$\omega(\vec{x},t) = \omega^*(\chi_{[\tau,t]}) \tag{2.37}$$

As the pullback from time $t = 0$ could involve hundreds or thousands of sub-maps, the computation of the fine vorticity in Hermite form is very expensive, especially with increasing amount of sub-maps to compose with. It is therefore only done for the initial condition of each sub-map. In order to resolve the fine scale structures that arise in the vorticity profile of turbulent flow, the fine grid to be used there should be chosen of larger size than that of the flow map or stream function.

2.3.5 Computation of Biot-Savart-Law in Fourier space

In order to advect the maps to the next time-step, an approximate of the velocity of the flow is needed to integrate along the characteristic curves. This can be computed using the Biot-Savart-Law from equation (2.12). By choosing the computational domain to be the 2-flat Torus $\mathbb{T}$ of size $[0, 2\pi]^2$, all quantities can be described by Fourier series. This allows to benefit from spectral derivatives in order to numerically compute equation (2.12). This drastically simplifies the computation of the inverse Laplacian and the derivatives, as these operations are merely just scalings with the

respective wave-numbers.
Instead of computing the velocity and using it directly, the stream function will be computed in Hermite form. From that, the velocity can be sampled at any arbitrary point with third order accuracy in space.
After transforming the vorticity into Fourier space with $\widehat{\omega}(\vec{k}, t_n) = \mathcal{F}(\omega(\vec{x}, t_n))$, the entries of the stream function can be computed in Fourier space via

$$\begin{aligned}
\widehat{\Psi}(\vec{k}, t_n) &= -\frac{1}{k_x^2 + k_y^2}\widehat{\omega}(\vec{k}, t_n) \\
\widehat{\partial_x \Psi}(\vec{k}, t_n) &= ik_x\widehat{\Psi}(\vec{k}, t_n) \\
\widehat{\partial_y \Psi}(\vec{k}, t_n) &= ik_y\widehat{\Psi}(\vec{k}, t_n) \\
\widehat{\partial_x \partial_y \Psi}(\vec{k}, t_n) &= -k_x k_y\widehat{\Psi}(\vec{k}, t_n)
\end{aligned}$$

In the case of $\vec{k} = \vec{0}$ the computation of the inverse Laplacian reduces to $\widehat{\Psi}(\vec{0}, t_n) = 0$. Thereafter, the values can be re-transformed to real space. Using this approach is not only fast, but also allows for accurate computation of the inverse Laplacian. Especially with strong gradients present in the vorticity, a finite difference approach would be challenging in terms of accuracy and feasibility.

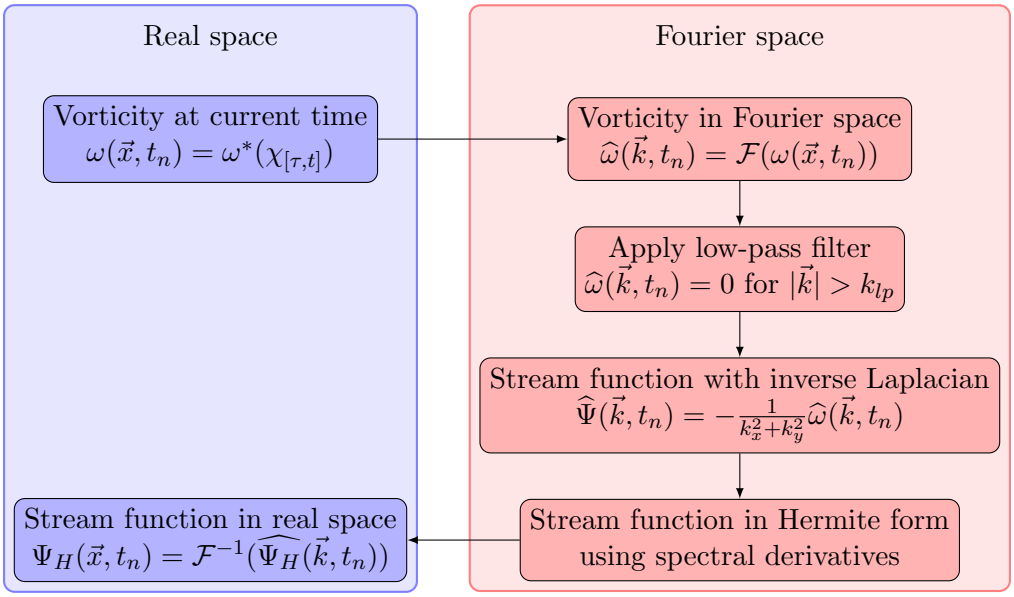

Figure 7: Flow-chart for the computation of the stream function in Hermite form within the final framework.

Another benefit of the use of computations in Fourier space is, that filters on the frequency can be easily applied. Especially low-pass filters are often used in resource-heavy fluid simulations in order to stabilise the solution. Possible filters

can and were implemented directly in the computation of the Biot-Savart law, which will be further explained later in the section (4.4.2). A flow-chart depicting how the stream function in Hermite form is computed from the vorticity and current sub-map can be found in the following figure 7.

2.3.6 Overview over the complete framework

All steps discussed in the previous sections together form the key framework in order to compute solutions to the 2D incompressible Euler equations using the characteristic mapping method. As several mathematical and numerical concepts have been explained, this section is dedicated to connecting everything together and forming a broad overview. All major building blocks and there connection and data dependency are depicted in the flow-chart in figure 8.

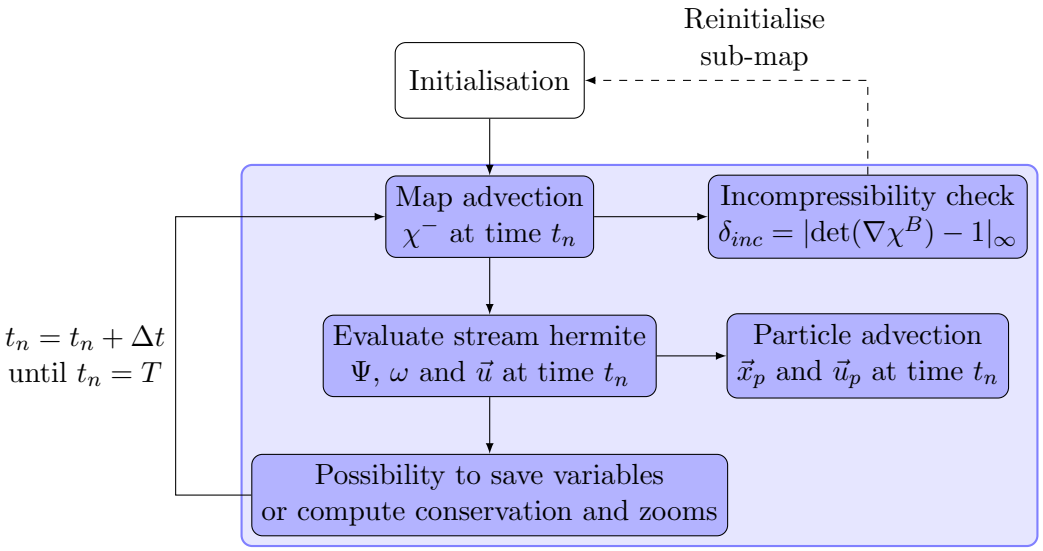

Figure 8: Flow-chart for the advancement of the complete framework.

A computation starts with an initialisation step. There the first sub-map is initialised as the identity map in Hermite form on the coarse grid. Thereafter the vorticity as the initial condition of this sub-map is sampled from a given analytical or discrete initial condition in Hermite form on a fine grid together with the stream function in Hermite form on the psi grid.
With that the actual simulation loop can start, where at first the flow map is advected. This is done by using the GALS-method where foot-points are advected by the velocity being interpolated as the derivative of the stream function. Using finite-differences the map is updated in Hermite form at the current time t_n. With the incompressibility condition the sub-map is tested if it exceeds a given thresh-hold. If that is the case a new sub-map will be initialised together with the vorticity as

the initial condition and is used from here on.
After that, the stream function is evaluated in Hermite form at the current time, which is also used for the advection of point particles embedded in the flow. At last, the variables can be saved or further quantities as the energy, enstrophy and palinstrophy computed.
This loop is repeated until the final time is reached. It is important to note that Δt as the time-step is not fixed but could be set arbitrarily in order to reach specific time targets within the simulation span.

3 Computational framework

As all numerical frameworks have to be eventually discretised and solved in many small cells or parts, the computing power is a key factor to limit the feasibility of large-scale computations. Therefore, not only is a mathematically and numerically efficient implementation of a physical problem important, but it has to be adapted in order to run well on modern processing units too. This comes with adapting it to the current world of central processing units (CPU) and memory management. A famous observation describing the growth of the computing power is the so-called "Moore's law", stating that the amount of transistors in integrated circuits double every approximately two years. As CPUs largely consist of those transistors, this can be seen as a limiting factor to the growth of computing power too. However, important problems of fluid dynamics and turbulence expect resources far beyond current standards for full direct numerical simulations (DNS). This is due to the fact, that 3D-turbulent simulations require a computation scaling with Re^3. It is estimated that DNS computations of a full aircraft at a Reynolds number of $Re = 10^7$ will not be achieved in the next 50 years with current growth of computational power. Also for solutions of the Euler equations this becomes a challenge, as the needed grid-size goes to infinity, so the highest resolution possible is pursued in order to minimise numerical approximation.
One huge step in the advancement of solving numerical equations is to increase the amount of computing cores to effectively solve the equations. For calculations with CPUs, this often means dividing a domain into smaller sub-domains, each individually solved by one core with memory exchange at the boundaries. However, the developed equations in the presented framework can benefit from more optimisation. In fact, three major aspects have to be computed during a time-step for the CM-method, being the Hermite interpolation, numerical advection of the flow map and Fourier transformations to compute the stream function. Luckily, all of those computations do not require many interactions between neighbouring cells to compute further quantities. This largely comes due to the fact, that the system of equations is governed by a transport equation as the central part, where each fluid particle can be individually advected. This stands in contrast to viscous flow, where movement is dependent on the density between neighbouring fluid particles.
Due to this largely independent character of the computational building blocks, the computing power of graphical processing units (GPUs) can be harnessed, where each computational core acts largely independent. In fact, the whole framework has been implemented to be solved entirely on GPUs. This was firstly done by Badal Yadav in the scope of his master thesis [19] in Montreal, Canada. The code was subsequently refined by Thibault Oujia and later Nicolas Saber in his master thesis [15] at Marseille, France.
In the scope of this work, the code received a complete major overhaul, aiming to improve the readability and transitioning it to more modern programming techniques. This chapter of the thesis aims to introduce important programming aspects when

dealing with GPU applications, mainly the Cuda framework. The introduction is in accordance with the current official programming guide, which can be found in [9]. In addition, a broad overview of major novel features regarding the code will be given to document the work done.

3.1 Introduction to Cuda

Cuda stands for "Compute Unified Device Architecture". With hardware drivers and an API to extend the programming language "C++", it gives a toolkit to enable the utilisation of GPUs in order to solve computational tasks efficiently. As those GPUs offer much higher instruction throughput and memory bandwidth than a CPU, this can lead to significant improvements in computation speed. Nevertheless, GPUs are designed fundamentally different, so implementing a task has to focus on other aspects. The biggest design difference is the execution strategy of several instruction sets. While the CPU commonly computes everything in sequence and synchronous, the GPU is designed to run thousands of different executions at once in parallel without any given order.

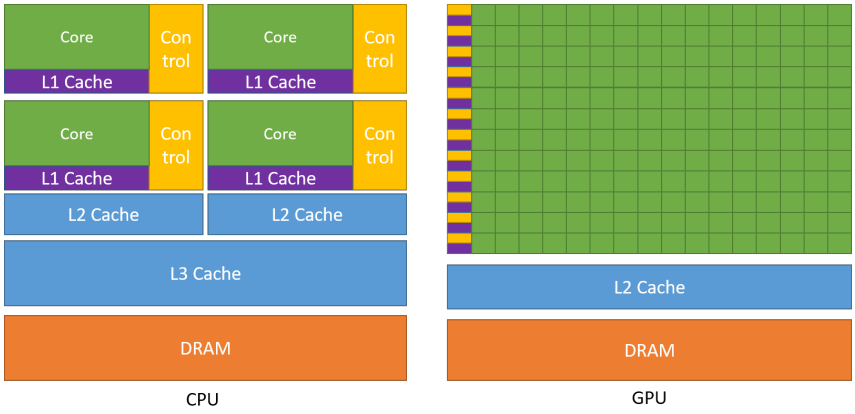

Figure 9: Different core structures between CPU and GPU hardware, [9]

As memory in CPU applications is mainly generously available due to the computation being the bottleneck of instruction speed, one usually does not have to pay attention to the transfer of data in computational tasks. However, for graphical processing units this is not the case and specific care has to be taken on the memory latency of presenting data to the cores for computation. This is also depicted in figure 9. While CPU cores usually feature their own caches and control units, individual cores of a GPU share local caches and control units.
In order to form a universal approach, CUDA introduces a general purpose programming model to maximise the usage of GPUs while making it possible to optimise the memory management in detail. This model aims to logically redefine executions on GPUs similar to CPU instructions, to greatly simplify the usage. The biggest

being the introduction of so-called “Kernels”. These kernels logically share great resemblance to normal programming functions, being composed of an in- and output. However, by definition they are functions being executed numerous times in parallel by different threads. While logically showing great resemblance, kernels introduce further new terms and specific ways to be executed. These will be presented in the following sections.

3.2 Structure of Cuda-kernel execution

All kernel executions are meant to be executed on a logical grid, which can be 1-, 2- or 3-dimensional. The given grid is divided into two subdivisions, being a block and thread, shown in figure 10. While a thread resembles the smallest structure and can be compared to a node on a grid of a computational domain, a block aims to bundle up several computations with logical cross-access to optimise parallel data transfer. Each thread can be individually identified by given block- and thread-ids, however, the order in which they are executed is not known beforehand. They are therefore completely independent of each other. Sharing data is realised on block-level. Blocks are computed individually by Streaming Multiprocessors (SMs). Those SMs can be logically roughly compared to CPU cores, where all blocks are computed independently from each other. A task can therefore easily be scaled dependent on the availability of SMs in the hardware.

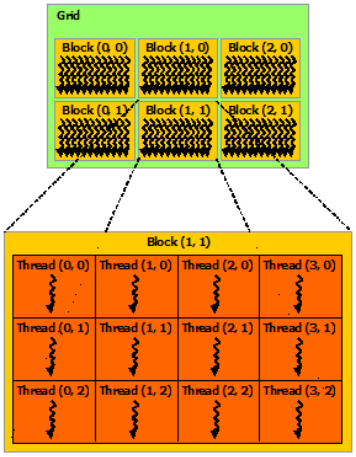

Figure 10: Subdivision of a grid in blocks and threads, [9]

The execution of a block is managed in a unique architecture called SIMT (Single-Instruction, Multiple-Thread). Several instructions are pipe-lined to leverage instruction-level parallelism within a single thread. Group of 32 concurrent threads form a warp, which are created, managed, scheduled and executed together. The term warp itself originates from weaving, the first parallel thread technology. They start at the same program address and only one common instruction is executed at a time. Due to this, full efficiency is only realised when all individual thread instruction counter addresses point to the same location. However, different code execution is still possible, with all other threads being disabled in the meantime. Due to the low occupancy, code with different paths within a warp are not recommended and can drastically effect the performance.
For example it is very common in numerical simulations to deal with boundary conditions depending on the position on the computational domain. However, this

could mean for a kernel implementation, that all threads within a warp have to wait for the threads located at the boundary to finish computing the boundary condition. Dependent on the code structure this can quickly lead to diverged instruction paths. Special care should therefore be taken in order to avoid path divergence within a Cuda kernel.

```
__global__ void diverging_periodic_derivative(double* x, double* dx, int N)
{
    int t_id = blockDim.x * blockIdx.x + threadIdx.x;  // id of current thread
    if (t_id == 0) {  // case for left boundary
        dx[0] = (x_in[1] - x_in[N-1]) / 2.0;
    }
    else if (int_id == N-1) {  // case for right boundary
        dx[N-1] = (x_in[0] - x_in[N-2]) / 2.0;
    }
    else {  // case inside domain
        dx[t_id] = (x_in[t_id+1] - x_in[t_id-1]) / 2.0;
    }
}
__global__ void conforming_periodic_derivative(double* x, double* dx, int N) {
    int t_id = blockDim.x * blockIdx.x + threadIdx.x;  // id of current thread
    // cases are connected by warping indices at borders back inside the domain
    dx[t_id] = (x_in[t_id+1 - N*(t_id==N)] - x_in[t_id-1+N*(t_id==0)]) / 2.0;
}
```

Code example 1: Two functioning Cuda Kernels to solve a numerical central difference scheme with periodic boundary conditions in a warp-divergent (top) and non-divergent way (bottom).

The previous code example 1 shows two kernel functions in order to compute a numerical derivative for an input vector of size N. While the first kernel takes special care to formulate the boundary conditions, this is only executed by one warp at a time with all the other warps waiting. The second kernel directly takes the boundary conditions into account by projecting the indices of boundary cells back into the domain with the help of a logical statement, wich for other parts of the domain reduces to zero. While for usual computations this would lead to unnecessary computations, it actually prevents a kernel execution from diverging paths.

3.3 Memory of GPU devices

Due to the high amount of processing cores inside of GPUs, the individual data cache per thread is small compared to CPU caches and threads cannot access each others private local memory. However, threads are connected in blocks, which have shared memory visible for all threads within the block. All threads from all blocks can access the same global memory. This leads to a three-level data hierarchy, which is also shown in figure 11. In addition, there exist two additional read-only memory spaces available by all threads, which is the constant and texture memory space.

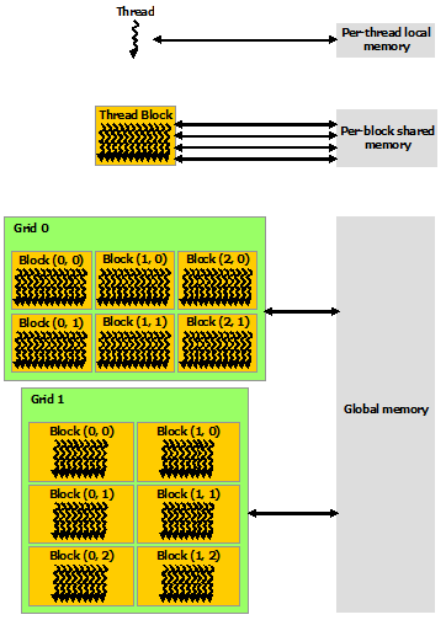

Figure 11: Main memory access of the different subdivisions, [9]

Similar to CPU architecture, the data transfer is faster, the closer the data is located to the core with the fastest being the local memory. This is the memory which is being passed by value as the input or created inside a function. It is only visible to the current thread and cannot be exchanged with neighbouring threads.
The second memory available is the shared memory. It gives rise to the possibility for inter-block communication of data and has to be defined and loaded for each kernel execution individually. It is especially useful if several threads load or save similar data at different positions, so that the data only has to be loaded once. For newer Cuda architectures with compute capabilities 6.x and upwards it resides in the same memory as the local memory cache, where the individual capacity has to be divided for those two memories. The developing and testing setup used in this work featured a Cuda architecture with compute capability 5.0. There, the shared memory has it's own fixed cache. This gives large differences in computation between those two architectures, as the local cache is one limiting factor of the current computations.
The third memory is the global memory. All kernel launches can access the same data residing in the global memory, which is loaded and stored into the local cache in order to modify data. This loading and storing process is especially costly. In addition, different threads cannot assign data to the same global address simultaneously. A function violating this behaviour can be found in the preceding kernel function:

```
__global__ void conflict_energy_sum(double& E, double* vel, int N) {
    int t_id = blockDim.x * blockIdx.x + threadIdx.x;  // id of current thread
    // E is reference to global memory leading to unexpected behaviour
    E += 0.25 * vel[t_id] * vel[t_id] / (double)N / (double)N;
}
```

Code example 2: A Cuda Kernels trying to compute the Energy from the velocity leading to memory conflicts.

Each thread in a warp first loads the value for E, then evaluates the assignment and later tries to assign the value to E. However, during this process another thread

could have already assigned a value to E, which is now overwritten. This problem can be circumvented in this case with atomic-functions, practically combining all steps from loading to saving into one instruction. The example illustrates the challenge of global memory handling within Cuda Kernels. The sample applies for shared memory as well, where each warp still individually deals with the instruction set of a function. All warps have to be synchronised in case any loading or storing has to be done by another thread not inside the same warp.
Constant memory has loading time comparably to global memory, however, it's value is always broadcasted to all threads inside a warp, making it optimal if all warps try to access the same data at the same position. For the use inside kernels it is optimal for constants, which do not change over the course of the kernel execution. This is often the case for numerical applications, as there are often many constant factors for numerical schemes involved.
The texture cache is optimised for memory reads with two-dimensional data, with higher read performance for data being located closely to each other in both dimensions.

3.4 Work on the code

Many kernels in the code have been optimised in the scope of this work. In fact, almost every line present was adapted to be more readable, accessible or accurate. The initial structure of the code was clearly written in a C-style behaviour. In the 50 years since the programming language C was first introduced in 1972 however, a lot has changed. While the reworked code still misses a lot of C++ like style, a lot of effort was invested into making the code more broad and to transition it to more objective-oriented programming procedures.
The code in the state of summer 2021 featured already a fully working implementation of the characteristic mapping method to solve the 2D incompressible Euler equations. However, large parts of it were uncommented, making the way in which it worked unclear. Also, usage for more refined simulations was complicated and different settings and parameters scattered along the whole structure. In the process to transition it to a version being usable by more people, the structure was reworked and widely commented. Repeated code with little to no modification, which is also called boilerplate code, was merged and old and redundant functions were removed. During this process the memory usage by temporary variables in the GPU global memory was reduced to 1/6th of its initial size. While before the GPU-memory available imposed major limits to the computations, it now makes up less of a bottleneck in comparison to the CPU RAM memory to store the individual sub-maps and the overall memory to save processed data.
The structure of the code-files was reworked and clustered into numerical-related, CMM-simulation-related and file and user related scripts. Together with a more clear and flexible compilation and assembly of the machine code, the overall file structure of the framework was improved significantly. The same applies to the in- and output of simulations. All settings have been coalesced into a settings-file, where they can

be easily altered and have a more detailed description available. Each simulation run can receive input on command-line execution, enabling automated scripting of larger batch-runs. In addition, each simulation can read in settings from a parameter-file and outputs it's setting to a parameter file, giving more transparency on the actual values being set also for later understanding. For the output, the frequency and variables to be saved can be easily altered inside of the parameters, enabling very detailed precision for wanted quantities without unnecessarily filling up the machine hard drive storage. Computations of individual zooms, samples at specific grid-sizes and particle computations can be easily en- or disabled and is stored in a reasonable file structure. Each computation also keeps track of the timing, incompressibility error and other quantities during the run and outputs them to the console if requested.

Overall, the structure of the code was changed from a purely single-user prototype to a code, that can be used and executed by several users without major knowledge of the code itself. It is also currently in the final stages of being made open-source on the platform GitHub. While science often features individual researchers advancing in previously unknown ideas, requiring often prototype code to be written, this is often hard to deal and exchange with into larger communities of other research groups. The rework of the code was made in order to be accessible by further scientists to come, even if it is just parts of the code to be reused.

However, also the current situation of the code still represents a work in progress. The biggest being a possible extension to three-dimensions. While this does not come with major numerical challenges, it does need some rethinking and reworking of the data structure in order to be implemented efficiently. Some other enhancements could be saving the data in HDF5-format, which represents a more modern and compact way to save the created data, or rethinking the velocity interpolation. Also, currently all different kernels are working synchronous, in order to benefit from several GPUs or to optimise data and computation handling at the same time, these have to be reworked to fit those needs.

Another big challenge is a data-efficient implementation of the Hermite interpolation. Currently, the loading of the data to be implemented from global memory is not optimised, leading to filled up data caches. This introduces major constrain on the speed of computation, with the data transfer rate becoming the major bottleneck in comparison to the computational speed. However, as the data read especially for map values being advected during the GALS-step is largely unpredictable for turbulent velocity fields, a more optimised and Cuda-like implementation was not yet achieved and could improve the computation times drastically.

4 Improved methods and convergence validation

After having presented the mathematical, numerical and computational backbone of the characteristic mapping method, further refined details in the scope of this work are presented in this chapter. Here, a strong focus on fine-tuning numerical parameters or methods is given. This aims to maximise the quality of produced results. First of all, various time-stepping schemes with different orders of convergence are introduced. The map update step for the GALS framework is presented in more detail and higher order finite differences schemes are introduced and their properties investigated. Next the actual orders of convergence of the code are validated in time and space, to confirm the theoretical properties of the method. At last in this chapter, the importance of minor settings for the method is discussed in order to assess their behaviour on improving the results or stabilising the simulation. This should aim to quantify the influence of changing different hyperparameters.

4.1 Addition of higher order time-stepping schemes for the GALS advection

The initial code developed by Badal Yadav in [19] featured a second order Adam-Bashford scheme to advect the characteristic flow map each time step. However, it was found that this implementation was not only disadvantageous in terms of accuracy, with a third order implementation being preferred to go along the third order Hermite implementation in space, but also featured a fixed point iteration, which imposed large computational costs.
Already in the work of Nicolas Saber in [15] , a third order Runge-Kutta method was presented and implemented, but not yet sufficiently validated. In the work of this thesis, more time-stepping schemes with different order of convergence and optimisation were introduced and will be presented in the following sections. Later, in section (4.3), those schemes will be validated. The schemes chosen are all in the nature of Runge-Kutta methods, being composed of one or several intermediate estimations and can be represented in so-called Butcher tables. The presented tables for the methods are taken or derived from [4].
Special caution has to be taken when reading the Butcher tables, as the backwards flow map operates backwards in time. A depiction of that can also be found in figure 4.

4.1.1 Lagrange interpolation of the velocity

The characteristic map is advected by the velocity of the flow, which is only given at the current time t_n. However, in all computations of the backwards map, the velocity is needed at least at time t_{n+1} and possibly further intermediate times too, where the velocity is not available. To overcome this issue, previous values of the velocity

are saved and values for upcoming time instants are extrapolated by using Lagrange polynomials. Using more previously computed velocity values increases the order of the Lagrange polynomials and therefore also the order of the extrapolation. The polynomials are constructed by

$$L_i(t) = \prod_{\substack{k=0 \\ k \neq i}}^{l} \left(\frac{t - t_k}{t_i - t_k} \right) \quad \text{with } i \in \mathbb{Z} \ , \ i \in [0, l] \tag{4.1}$$

where l is the order of the Lagrange interpolation and L_i is the resulting weighting factor. The velocity at any time t can then be inter- or extrapolated with

$$\tilde{u}(x_n, t) = \sum_{i=0}^{l-1} \Big(L_i(t) \cdot u(x_n, t_{n-i}) \Big) \tag{4.2}$$

Due to the flexible implementation, the time instants of the Lagrange polynomials can be chosen arbitrarily. As a consequence, the length of a time-step is not fixed by the polynomials and can be altered during the simulation. With this, modern approaches like Runge-Kutta-Fehlberg methods or computing at specific time instants can be implemented. However, if the time-step is increased too largely, the Lagrange polynomials cannot extrapolate the velocity well. Usually, Lagrange interpolation can show oscillations quickly, as a finite amount of polynomials cannot interpolate exponential behaviour well. However, this issue was not observed with a maximum of fourth order interpolation used.

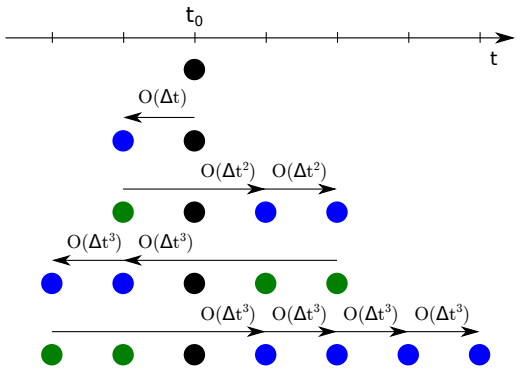

Figure 12: Initialisation of the velocity by computing with increasing orders of Lagrange interpolation for a third order simulation.

As the intermediate velocity is a superposition of velocities satisfying the incompressibility condition (2.3) , the intermediate velocity therefore is also divergence free:

$$\nabla \cdot \tilde{u}(x_n, t) = \sum_{i} \Big(L_i(t) \cdot \nabla u(x_n, t_{n-i}) \Big) = 0 \tag{4.3}$$

At the beginning of the simulation the values for previous velocities are not given. In order to use the interpolation from the first step on-wards, these values have to be initialised. An easy idea to initialise those values would be to compute a few steps backwards in time with a first order method, where no interpolation is needed. However, a more sophisticated way was chosen, as initialisation errors are easily

propagated but only account for a small fraction of the computation time for long simulations.
First of all one first order step is computed to advect the map and the velocity is computed at that time-step. Afterwards the map is reinitialised and the direction of computation inverted. Now, the velocity at time t_0 and the first order estimation for the previous time-step is available and can be used with a second order time-stepping method. With that two steps are computed to compute the map advection and velocity and afterwards the map is again reinitialised and the direction of computation inverted. This procedure can be continued, until all needed previous velocities have been computed with a time-stepping method of the requested order for the computation. An illustration for this can be found in figure 12. Here, the computation is always started from time t_0 (in black) to compute new time-steps (in blue). With increasing order of the time-stepping methods more previous values are needed, given in green. When all previous velocities were computed with a method of the desired convergence order, the simulation can start.

4.1.2 Presentation of different classical schemes of first to fourth order

Here, all the new time-stepping schemes are presented with their respective Butcher tables. In total, six Runge-Kutta methods are now present in the framework, where the explicit Euler and the third order classical Runge-Kutta method were taken over from previous work.
In order to advance the flow map to the next time-step, the characteristics have to be advected to the next time steps in equations (2.31) and (2.32). This will be done by the Runge-Kutta methods with intermediate time-steps k_j:

$$\chi_{[t,t+\delta t]} = \vec{x} + \Delta t \sum_j b_j k_j^+ \tag{4.4}$$

$$\chi_{[t+\delta t,t]} = \vec{x} - \Delta t \sum_j b_j k_j^- \tag{4.5}$$

Dependent on the evaluation of the forward or backwards map, the computation directions changes as well. The amount of estimations $j \in \mathbb{Z}, j \in [1, n]$ is dependent on the convergence order n of the method. The factors a_{mj}, b_j and c_j can be extracted from the Butcher table. Further explanation can be found in [15] or [4]. Each intermediate evaluation is computed by

$$k_j^+ = \vec{u}(\vec{x} + \Delta t \sum_{m=1}^{j-1} a_{jm} k_m, t + c_j \Delta t) \tag{4.6}$$

$$k_j^- = \vec{u}(\vec{x} - \Delta t \sum_{m=1}^{j-1} a_{jm} k_m, t + \Delta t(1 - c_j)) \tag{4.7}$$

The already included first order implementation constitutes to the Euler explicit method (EulerExp). It's Butcher table is fairly simple ([4] p.185):

0	
	1

It only consists of updating the map position using the current position. Since only one function evaluation is done, no intermediate steps actually have to be constructed.
The second order method was chosen as the Heuns method and uses the second order stencil from ([4] p.185) with $c_2 = 1$ for the second step. The Butcher table for this method is:

0		
1	1	
	1/2	1/2

Due to the equal averaging applied in the final evaluation, it is also known as the explicit trapezoidal rule.
For the third order method, a classical third order implementation was chosen (RK3). This constitutes to the general third order case I from ([4] p.186) with $c_2 = 1/2$ and $c_3 = 1$. It has been already introduced in the work of [15] and is included here for completeness.

0			
1/2	1/2		
1	-1	2	
	1/6	2/3	1/6

The scheme chosen for the fourth order implementation is also very well known and widely applied to different applications, due to its simplicity. It is the classical fourth order Runge-Kutta method (RK4), which is given in ([4] p.194) as the Case V with $b_3 = 1/3$. The Butcher table for this method is:

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
	1/6	1/3	1/3	1/6

While an implementation of third order was favoured for the framework, the fourth order implementations where straight forward in their implementation and can help to assess the quality of a third order implementation.

4.1.3 Remark on the computational complexity and construction of more optimised third and fourth order schemes

When computing the velocity for the Runge-Kutta methods, they have to be interpolated in space with the Hermite interpolation in order to be located at the right positions. While each interpolation should feature one estimation of the velocity for

each spatial dimension, this has to be done several times in the Lagrange interpolation. In fact, a Lagrange interpolation of order l means, that l velocities have to be interpolated, effectively heavily increasing the computational cost with increasing order of interpolation.
In order to deal with this, higher order Runge-Kutta methods can be chosen with more optimal time-stepping instants chosen. If they coincide with times at which the previous value is available, no interpolation has to be done. However, while this reduces the computational cost, it also reduces the accuracy of the Runge-Kutta methods, as evaluations further away reduce the accuracy at which the time evolution of the velocity is predicted by the intermediate evaluations. However, as they still contain the same order of convergence, an implementation might be favourable and the efficiency will be further investigated after the convergence validation in section (4.3.3).
For both the third and fourth order method a different Butcher table was chosen. As the forward and backwards flow map point into different directions, the implementation for both differs. The third order method with the general case I from ([4] p.186) can be chosen with different time instants outside of the integration interval $[t_n, t_{n+1}]$, making it optimal to minimise the amount of Lagrange interpolations For the backwards flow map the coefficients $c_2 = 1$ and $c_3 = 2$ were chosen, giving rise to the Butcher table (RK3Mod):

0			
1	1		
2	4	-2	
	5/12	2/3	-1/12

The time instants $c_2 = 1$ and $c_3 = 2$ constitute for t_n and t_{n-1}, where the velocity value is available. Therefore, only for the first evaluation at t_{n+1} a Lagrange interpolation has to be done. This reduces the amount of Lagrange interpolation needed from two to one.
For the forward flow map $c_2 = -1$ and $c_3 = -2$ were chosen:

0			
-1	-1		
-2	-8/5	-2/5	
	23/12	-4/3	5/12

With the velocity at times t_n, t_{n-1} and t_{n-2} all available, no Lagrange interpolation has to be done.
For the fourth order method all available cases have time instants in the interval $[t_n, t_{n+1}]$. The only way to reduce the amount of Lagrange interpolations is therefore to choose methods with as many evaluations as possible at time t_n. Luckily, there exist methods with two evaluations at that time which can be used for the backwards and forward flow map, which are case IV with $b_4 = 1/4$ for the backwards flow map (left) and case III with $b_3 = 1/6$ for the forward flow map (right) from ([4] p.193) (RK4Mod):

0				
1	1			
1/2	3/8	1/8		
1	0	-1/3	4/3	
	1/6	-1/12	2/3	1/4

0				
1/2	1/2			
0	-1/2	1/2		
1	-3/2	3/2	1	
	0	2/3	1/6	1/6

The coefficient b was chosen arbitrarily to vanish one term. For both the forward and backwards flow map computations, one Lagrange interpolation is saved, resulting in three less Hermite interpolations of the velocity.

Abbreviation of Method	Lagrange interpolations backwards	Hermite interpolations backwards	Lagrange interpolations forward	Hermite interpolations forward
EulerExp	1	$2l$	0	2
Heun	1	$2l+2$	1	$2l+2$
RK3	2	$4l+2$	2	$4l+2$
RK3Mod	1	$2l+4$	0	6
RK4	3	$6l+2$	3	$6l+2$
RK4Mod	2	$4l+4$	2	$4l+4$

Table 1: Amount of Lagrange interpolations and Hermite interpolations of velocity values for each time-stepping method.

As can be excerpted from table 1, the two different third order Runge-Kutta methods with Lagrange interpolation in respective order need different amounts of interpolation. For the backwards map, 14 Hermite interpolations are needed for RK3 and 10 for the modified version RK3Mod, reducing the computational complexity by 28.5%. For the forward map it is even more extreme with still 14 interpolations for RK3 and only 6 Hermite interpolation for RK3Mod, reducing the computational complexity by 57.1%. For the fourth order Runge-Kutte methodes, the amount of Hermite interpolations reduces from 26 for RK4 to 20 for RK4Mod, needing 23% less evaluations.

4.2 Increased update order of the flow map update step

As the map is stored in bi-cubic Hermite form, the x-, y- and cross-derivative have to be updated in each step as well. For this the GALS framework is used, where foot-points close to the grid-points are chosen, advected and used to compute all new quantities, as it was already presented in section (2.3.3). The computation of the quantities are afterwards done by finite difference.
While previously only a second order stencil was implemented, this was updated to be able to choose a fourth and sixth order stencil too. Already for fourth order the error should be in the same order as the cubic Hermite interpolation, however, the sixth order was chosen to be implemented to check this expected behaviour.
The different stencil coefficients were computed by solving for the coefficients in

the Taylor expansion. The coefficients them-self were computed numerically solving the system of linear equations.This also enabled to easily check other stencils too. However, the cross-shaped stencils proofs to be the most efficient in terms of points needed per order.
The coefficients which have been computed for second, fourth and sixth order can be found in the following table.

Order	c_1	c_2	c_3
2nd	$\frac{1}{4}$		
4th	$\frac{1}{3}$	$-\frac{1}{12}$	
6th	$\frac{3}{8}$	$-\frac{3}{20}$	$\frac{1}{40}$

Table 2: Coefficient of different orders for the map update finite difference stencils.

Following, several simulations have been done with different stencil distances ϵ_m for second, fourth and sixth order stencils. All other simulation parameters were taken similar to the tests from section (4.3). During the simulations, the incompressibility error from equation (2.33) and enstrophy conservation error to the initial enstrophy from equation (4.12) where monitored in order to assess the properties of each implementation.
As it can be seen in figure (13), the previously used second order stencil is not able to compute the Hermite form of the flow map well. Not only does it randomly oscillate for small ϵ_m, but also has increasing incompressibility error for larger ϵ_m. An optimum for the incompressibility error can be seen at at around $\epsilon_m = 5 \cdot 10^{-5}$.

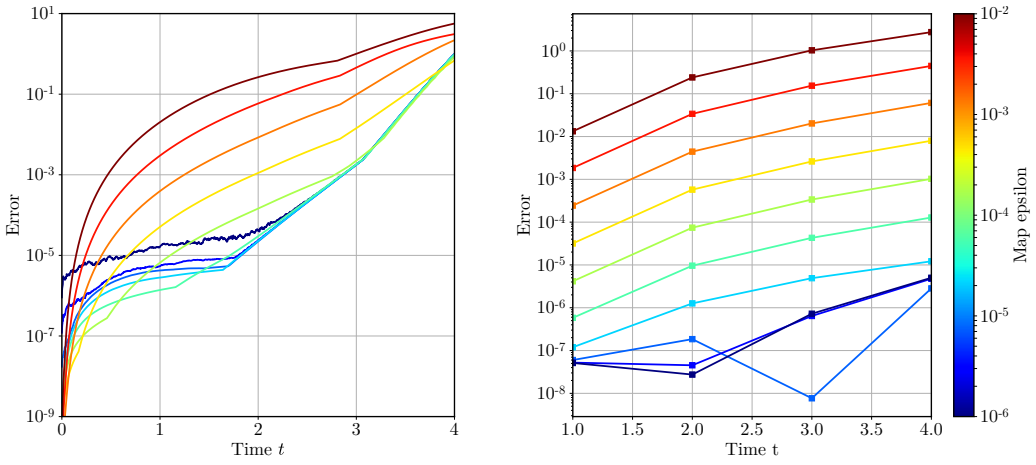

Figure 13: Incompressibility error (left) and Enstrophy conservation error in L_2-norm over time for different map epsilon ϵ_m with a second order map update stencil.

The right image of figure 13 shows the enstrophy conservation error to the initial enstrophy. While this error converges for smaller ϵ_m, a saturation is only reached

for values with oscillations for the incompressibility error. Overall, the second order stencil for the map update is not fit for the framework.

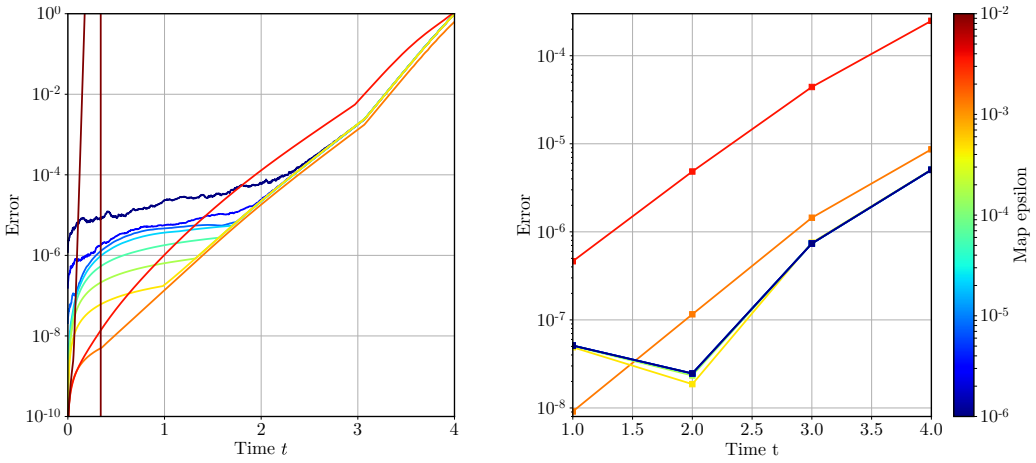

Figure 14: Incompressibility error (left) and Enstrophy conservation error in L_2-norm over time for different map epsilon ϵ_m with a fourth order map update stencil. Curves for low ϵ_m-values are overlapping for the conservation of enstrophy.

The fourth order stencil achieves more accurate result. With higher stencil distance, the incompressibility error is lowered and finally saturates almost completely for $\epsilon_m = 10^{-3}$. Even higher stencil distances diverge completely, which hints on the stability of the method. After longer time, all computations with different ϵ_m eventually cap at the same value. Eventually, the introduced error of the high frequencies of the velocity field takes over and dominates the growth of the incompressibility error. Different simulations showed, that this error is neither dependent on the time-stepping scheme, nor the step-size or the grid-size of the stream function to represent the velocity. It does correlate however with the growth of palinstrophy, which are strongly tied to the map gradients. A look at the conservation of enstrophy shows, that all different simulations cap, meaning that the influence of the map update step is negligible to the flow map computation and the fourth order stencil indeed improves the simulation. An optimal value was found around $1\text{-}5 \cdot 10^{-3}$. Surprisingly, this coincides with the final value used in [22]. There, an error estimate was given by the formula

$$E_m \approx \epsilon_m^4 + \delta_m \sum_{i=0}^{3} \left(\Delta x^i \epsilon_m^{-i} \right) \tag{4.8}$$

Where E_m is the L^∞-error being introduced by the map update step and $\delta_m \approx 2.22 \cdot 10^{-16}$ is the machine precision, computed in [15]. However, when using this formula, an optimum of $3 \cdot 10^{-4}$ for $\Delta t = 1/1024$ should be given, which could not be confirmed by the simulations using our framework.

The sixth order results, depicted in figure 15, do not show any major improvements over the fourth order stencil with only more improvements in stabilisation for higher ϵ_m. For the computation, four additional points have to be advected in comparison to the fourth order method. As is noticeably more costly, a usage is not recommended.

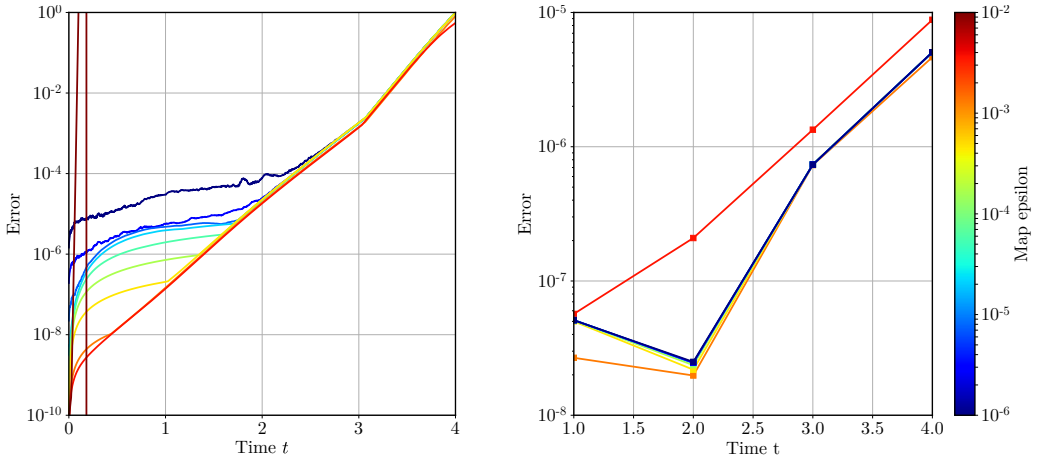

Figure 15: Incompressibility error (left) and Enstrophy conservation error in L_2-norm over time for different map epsilon ϵ_m with a sixth order map update stencil. Curves for low ϵ_m-values are again overlapping for the conservation of enstrophy.

4.3 Validation of convergence order in space and time

After all parts of the framework have been presented, their capabilities have to be validated to ensure, that the stated convergence properties are achieved. All convergence tests here have been done similar to [21] in order to compare the results with each other. Simulations have been done to verify an unchanged convergence behaviour in space, validate the convergence properties of the different time-stepping methods and to reconfirm the influence of different orders of Lagrange interpolation.

Name	Value	Name	Value
N_{coarse}	1024	N_{fine}	1024
N_ψ	2048	N_ω	1024
h_{fluid}	1/512		
ϵ_m	10^{-3}	Initial condition	4-mode-flow
Fluid time scheme	RK3	Map update stencil	4th order

Table 3: Settings for reference simulation, similar to [21].

In the preceding table 3, all important parameters for the reference simulations can be retrieved. No low-pass filter was implemented for the computations. The computed map underwent no remapping during the whole simulation time-span in

order to capture the error completely. All simulations are run until a final time of $t = 1$.
The error of four quantities were computed to validate the convergence order. Those are the flow map and vorticity error in L_∞-norm and the Energy and Enstrophy conservation error in L_2-norm. All the errors were evaluated by sampling the map on a uniform 2048^2-grid and computing the vorticity and velocity respectively on this map.

$$\text{Map error} = ||\chi_{ref}(\cdot, t_n) - \chi(\cdot, t_n)||_\infty \tag{4.9}$$
$$\text{Vorticity error} = ||\omega_{ref}(\cdot, t_n) - \omega(\cdot, t_n)||_\infty \tag{4.10}$$
$$\text{Energy error} = ||\omega(\cdot, t_n)^2||_2 - ||\omega(\cdot, t_0)^2||_2 \tag{4.11}$$
$$\text{Enstrophy error} = ||\vec{u}(\cdot, t_n)^2||_2 - ||\vec{u}(\cdot, t_0)^2||_2 \tag{4.12}$$

According to [21], an error bound for the characteristic map is given by

$$\tilde{\mathcal{E}}^n = \mathcal{O}\left(\Delta x^2 min(\Delta t, \Delta x^2 \Delta t^{-1}) + \Delta t^s + \Delta t^p\right) \tag{4.13}$$

Here s and p are the orders of the s-stage Runge-Kutta scheme and the used order of Lagrange interpolation for the velocity respectively.

4.3.1 Convergence order in space

As no feature concerning the spatial resolution changed in regards to the implementation of [21], the convergence order in space should remain unchanged. However, as the complete code of the framework around it was reworked, it is important to validate this property once more. Due to the bicubic Hermite interpolation used, errors of order $\mathcal{O}(\Delta x^3)$ are expected, which comes from the velocity to advect the map using the GALS framework. The velocity is sampled as the derivative of the stream function in Hermite form and is therefore present in third order accuracy in space. Simulations were run with different grid-size $\Delta x \in \{1/32, 1/64, 1/128, 1/256, 1/512, 1/1024\}$ with the computation of the smallest grid-size used as reference.
All four quantities show the expected third order behaviour, as can be concluded from figure 16, therefore validating the spatial implementation.

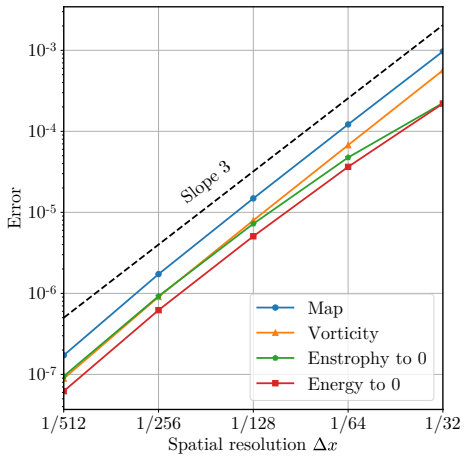

Figure 16: Convergence errors in space with a theoretical perfect third order convergence as reference.

4.3.2 Convergence order in time

With the introduction of new time-stepping schemes, the convergence behaviour in time was made largely flexible. However, as no new method to discretise or solve the flow map in time was used, the overall error estimates were not changed in regards to [21]. According to them, the estimate is directly dependent of the Runge-Kutta scheme and order of Lagrange polynomials being used. Due to there being six different schemes available, a majority of the graphs can be found in the appendix. For illustration, the error estimates will be shown here for the classical third order Runge-Kutta scheme with second and third order Lagrange polynomials to match the reference paper. Detailed convergence validation can be found in the appendix in section (9.1) and (9.2). For each method, simulations were run with $\Delta t \in \{1/8, 1/16, 1/32, 1/64, 1/128, 1/256, 1/512\}$, where the computation with lowest time-step was used as reference to compute the map and vorticity error.

The achieved results show the exact same behaviour as in [21]. While the conservation of energy and the error for the backwards flow map and vorticity are consistent with the minimum of the order of Lagrange interpolation and order of Runge-Kutta scheme, the Enstrophy only depends on the time-stepping scheme.

In terms of error quantity, the current framework achieve better vorticity error and worse conservation of energy. This, however, could be originating from the definition of the computation of those quantities, as the exact method of computing those quantities in the reference is not clear. The energy in the current framework is computed directly from the velocity, being the derivatives of the stream function. However, one could also sample it from the present stream function in Hermite form. The same applies for the vorticity, which in this case was sampled directly using the map from the initial condition, making it more precise than sampling it from the fine vorticity initial condition of the sub-map. As all quantities are of roughly the same

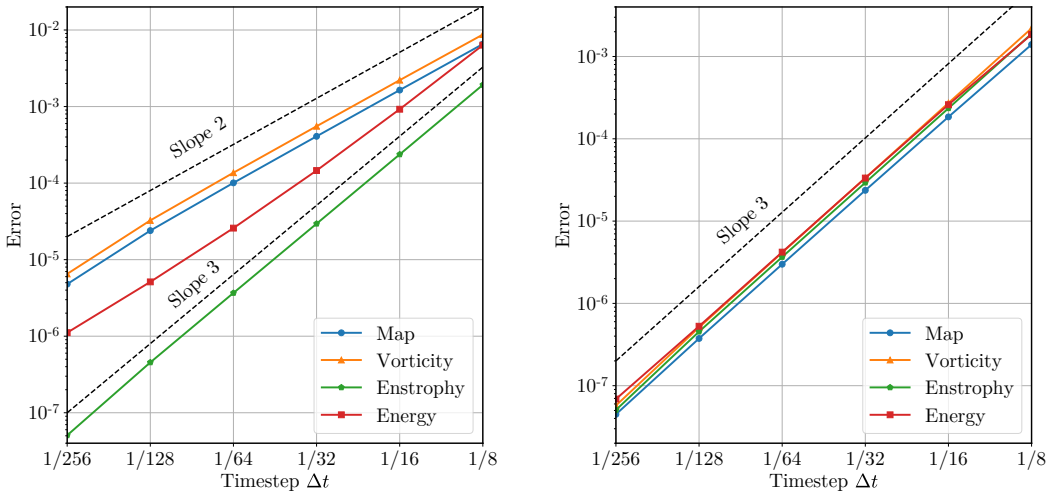

Figure 17: Convergence errors in time with second order (left) and third order (right) Lagrange interpolation and third order classical Runge-Kutta scheme.

order with maximum values ranging between $5 \cdot 10^0$ and $3 \cdot 10^1$, it is reasonable that all error quantities for a fully third order implementation are close to one another. However, as can be seen in the appendix, this is only the case for this specific setup. Nevertheless, the implemented schemes and methods work as intended.

4.3.3 Evaluation of efficiency for different time-stepping methods

After evaluation of the accuracy of the time-stepping methods together with the convergence in time, a remark will be given for the efficiency of the methods. Using the results from the convergence validation, the timings can be compared and possible stochastic effects between different simulations mitigated by averaging the timings. The variance can be used in order to quantify these effects.

$$\sigma_t = \sqrt{\sum_{h_i} (\tilde{t}_{C,h_i} - \tilde{t}_{C,mean})^2 \cdot \frac{h_i}{\sum h_i}} \tag{4.14}$$

Here $\tilde{t}_{C,h_i} = \frac{t_C}{h_i}$ is the average computational time per step of one specific computation and $\tilde{t}_{C,mean} = \frac{\sum t_C}{\sum h_i}$ is the total average time per step of all computations for one specific method. The second factor weights each computation dependent on their contribution to the total steps computed. The measurements were taken on a machine featuring an Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz with 8 logical CPUs and 16GB of RAM, paired with a NVIDIA GeForce 845M with 2GB of video memory (VRAM).
A low relative variance in comparison to the relative timings indicates, that the stochastic effects can be mitigated and the results of the different sets of simulations be compared against each other. The average computation time per step for each

time-stepping method is shown in table 4 together with the relative variance and relative timings in comparison to the first order Euler explicit method.

Name	EulerExp	Heun	RK3	RK3 Mod	RK4	RK4 Mod
$\tilde{t}_{C,mean}$ [s]	0.38	0.557	0.909	0.797	1.713	1.394
$\sigma_t/\tilde{t}_{C,mean}$	0.001	0.006	0.022	0.021	0.02	0.026
$\tilde{t}_{C,mean,rel}$	1	1.463	2.39	2.095	4.5	3.66

Table 4: Time per step, variance and relative time to EulerExp for computations with different time-stepping methods.

The low relative variances in the percentage range indicate that the stochastic effects being present for each set of computations can be mitigated. This is sufficient to compare the results between the different time-stepping methods against each other. The needed computation time is increasing with increasing convergence order, with especially the third and fourth order method having noticeably more computational effort. This comes from the increased amount of Hermite interpolations needed, as explained in the previous section.
By plotting the achieved computation time over the achieved convergence error, the efficiency of the methods can be illustrated, where lower error and timing results to a more efficient method. The plots for all different errors defined in section (4.3) is found in figure 18. It can be easily seen how the first order method is not efficient at all in comparison to the other methods. The second, third and fourth order methods are of surprisingly similar efficiency and only with smaller time-steps they start to differ. The modified versions for the third and fourth order time-stepping methods exceed their classical version for all different errors. However, also the saturation effects can clearly be seen for very low time-steps. This has to be taken into account when running simulations with very small time-steps in comparison to the step-size, as lower order methods would be highly more efficient once the error is dependent on different factors. All in all the third order methods proof to be nicely accurate for their given computational effort with the modified version to be chosen over the classical variant for simulations aiming to achieve optimal efficiency.

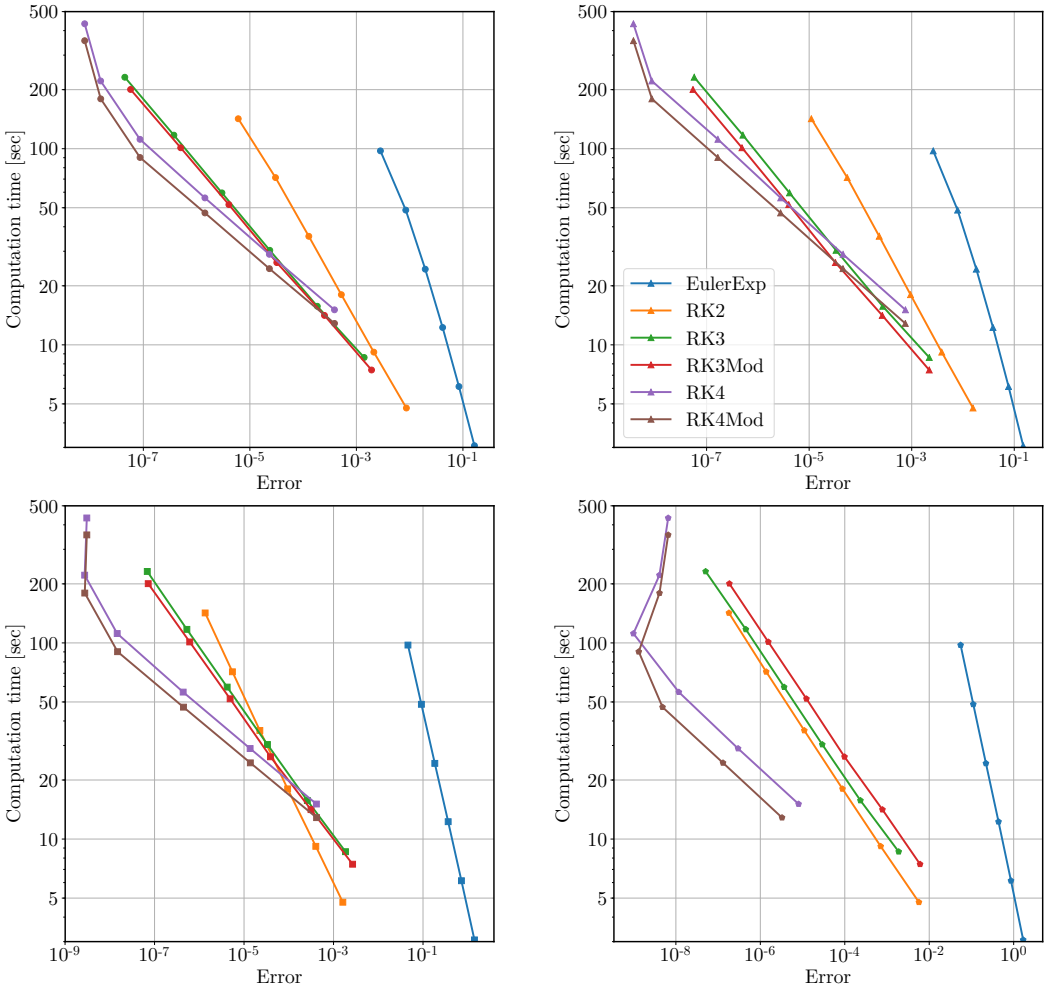

Figure 18: Computational time over achieved map (top left), vorticity (top right), energy (bottom left) and enstrophy (bottom right) error for different time-stepping methods.

4.4 Hyper-parameters to fine-tune balance between computational complexity and accuracy

Solutions of the incompressible Euler equations experience a transfer of enstrophy into finer scales over time , also called the energy cascade ([10], p.183). As the framework deals with a semi-Lagrangian approach, it still suffers the fact, that a finite uniform grid to represent the stream function can only capture frequencies until the Nyquist-frequency. When the amount of energy or enstrophy rises in finer scales, numerical errors start to occur. Vorticity present in finer frequency will be misinterpreted and aliasing effects occur, leading to a non-physical accumulation of enstrophy at fine scales close to the Nyquist-frequency.

While this does not only create a non-physical behaviour, it additionally imposes problems for the Runge-Kutta schemes to fulfil the incompressibility condition, as higher frequencies create steeper gradients inside the velocity field itself. With higher gradients, the numerical deviation to true incompressibility rises, which is implied on the flow map after map advection. Therefore, the incompressibility error from equation (2.29) is increased and more and more remappings are needed to keep the flow incompressible.
As each remapping comes with the computation of the very expensive vorticity on the fine grid from equation (2.36) and also increases the memory usage, this wants to be minimised, largely by dampening the influence of high frequencies while still preserving the information carried by the velocity.

4.4.1 Influence of different grid-sizes

With the core structure of the Characteristic Mapping method comes two different grid-sizes to capture the fluid dynamics. The first is the grid size of the flow map, also called coarse grid, which describes the uniform grid, to or from which the flow map will point to. As all sub-maps are applied in sequence, it is only important that a specific sub-map can contain all data needed to stay volume preserving. This is largely influenced by the composition of the stream function in Hermite form used to advect the flow map. Especially as the velocity to advect the flow is not smooth around cell corners when being sampled from the stream function, the volume preserving property of the flow map is largely dependent on it. As the volume preserving quantity is given by the incompressibility thresh-hold, an optimum of flow map size and amount of sub-map can be tuned for a given computation dependent on the available memory. With increasing map size, the total memory used by all sub-maps increase. The map size therefore has to be reduced when reaching CPU RAM memory limits, as all sub-maps are saved on the CPU RAM.
On the other hand, each sub-map samples the vorticity on a finer grid to be used as the initial condition. The aim there is to capture as many characteristics as possible. With exponentially rising palinstrophy for turbulent flow and shift of enstrophy to finer scales, initial vorticity in Hermite form on a sub-map will eventually not be able to capture all scales for longer turbulent simulations. In order to capture more scales, the size is usually higher than the size of the flow map. Larger fine grid increases the size of all temporary arrays used in the framework too, eventually filling up the GPU RAM. This effectively limits the largest available fine grid for each setup.
With those two different grid-sizes also comes a gap which has to be passed when evaluating the Biot-Savart-Law. Typically, the grid-size has to be reduced from the fine initial vorticity in order to be applied to advect the flow map. Over the course of this and previous work, a few techniques and ideas have been developed in order to achieve optimal results. This section elaborates on the idea of intermediate grids-sizes and gives a suggestion on how to set them during simulation runs.
The first idea is tied up with another idea being presented in the next section. There,

the idea of mollification is presented in order to reduce the grid-size at which the vorticity is sampled for the computation of the Biot-Savart-Law. This was done in order to reduce the load of the forwards Fourier transformation to compute the vorticity in Fourier space. This grid-size, in the framework dealt with as the "vorticity grid", however currently serves only little purpose. This is due to the fact, that the mollifier was replaced with filtering in Fourier space. In addition, the forward Fourier transformation is not as computation heavy at all in comparison to other parts of the code. A forward Fourier transformation on the full fine grid is therefore possible without heavy constraints on the computation. In simulation applications, the vorticity grid-size can therefore be set equal to the fine grid-size without any problems.
A more important grid-size however is the size of the stream function. Effectively, this decides the smoothness of the velocity to advect the flow map. It therefore directly influences the quality of the map advection. It was argued in [21], that this smoothness is of great importance. To feature a larger grid with more information available than present can be easily achieved by zero-padding in Fourier space by artificially increasing the domain with higher wavelengths with amplitude of zero. When computing the inverse transformation, this samples the stream function on a finer grid, smoothing out the spatial representation. In the scope of their work the grid for the stream function, called "psi grid" with parameter N_Ψ, even exceeded the grid-size of the fine grid. However, in the current framework this comes with several issues. First of all, the grid-size of the stream function actually imposes huge computational constraints on the time-stepping methods, due to the current implementation in Cuda eventually reaching memory caching limits. Additionally, with a psi grid exceeding the fine grid as the largest grid size, all auxiliary grids to compute the Fourier operations have to be increased as well and overall increases the memory usage of the GPU RAM.

A series of simulations have been done in order to assess the influence of the psi grid. Those were carried out with the settings of the reference simulation from section (4.3). Seven different simulations were run with increasing $N_\Psi \in \{1024, 1216, 1448, 1720, 2048, 2432, 2896\}$. Over the course of the simulation, the map error from section (4.3) and the incompressibility error from equation 2.33 have been monitored. The simulation with $N_\Psi = 2896$ has been used as a reference simulation in order to compute the map error in L_∞-Norm. The results can be found in figure 19. The graphs for the incompressiblity error show no significant improvement with only differences for small times, after which other effects start to dominate. A larger psi grid is therefore not beneficial to minimise the amount of remappings. However, the map error is reduced for larger N_Ψ. Overall, an error reduction of a factor 10 can be achieved in reference to the finest computation before a saturation effect is observed. Timings for the different computations can be found in table 5 with $\tilde{t}_{C,mean}$ being the mean time it took for one time-step to be computed on the used machine. The measurements were taken on a machine featuring an Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz with 8 logical CPUs and

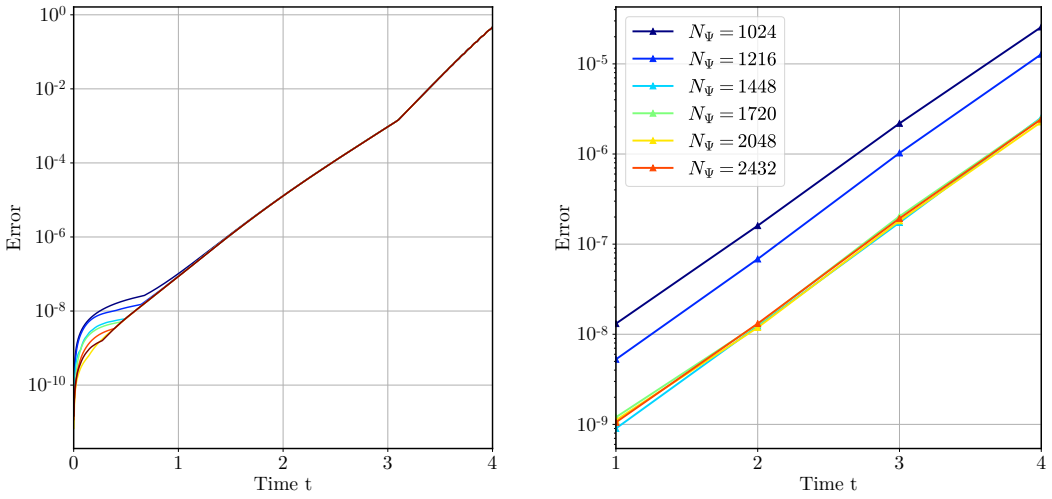

Figure 19: Incompressibility error (left) and flow map error (right) for computations with different N_Ψ.

16GB of RAM, paired with a NVIDIA GeForce 845M with 2GB of video memory (VRAM). The timings were not repeated and therefore still contain random deviations by the computation structure of the GPU. They are merely shown here to give a relative comparison between the methods. While for each method the main difference is the increased size of Fourier transformations, only the time-stepping methods take up more and more execution time with increasing N_Ψ, according to the debugging tools from NVIDIA. This comes from the fact, that for large N_Ψ the Hermite interpolation of the velocity for the Runge-Kutta methods has to load in more points from the Stream function in Hermite representation. This eventually exceeds the memory cache and creates a bottleneck, as all data have to be loaded in from the global memory, which is slow in comparison to cached data. This effect becomes extremely visible with an N_Ψ of two times the size of the coarse grid, with the whole computation taking up twice the amount of time, but also starts to be visible for a $N_\Psi = 1448$ with a 20% increase. While a decrease in the computed error of the flow map is desirable and can be shown with the computations, it also comes with sharply increased computational cost. However, nevertheless an increase of the psi grid to a certain degree actually benefits the computation. In the current framework it is yet still not recommended, as the effect is negligible in comparison to its increased computational complexity and memory requirement. Increasing the fine grid or order of times-stepping method more efficiently increase the accuracy of simulations.

N_Ψ	1024	1216	1448	1720	2048	2432	2896
$\tilde{t}_{C,mean}$ [s]	0.73	0.7	0.86	0.89	0.96	1.44	2.34
$\tilde{t}_{C,mean,rel}$	1	0.96	1.18	1.23	1.32	1.99	3.23

Table 5: Timings for simulations with different N_Ψ

4.4.2 Low-pass filtering of vorticity

In order to limit the gradients of the velocity, high frequencies can be filtered out. As in physical fluid flow only little energy is present in high frequencies due to the energy power law with exponent $k = -3$ ([10], p.183), the filtering does not discard much of the energy. It rather acts as an artificial dissipation for the map deformation.
In the previous work of [21], the filtration has been done by mollification. Before applying the Fourier transformation to compute the stream function in Hermite form as described in section (2.3.5), the vorticity was already sampled in lower resolution. The effect of a small cluster of points are convoluted with a smooth mollifier function in order to compute the down-sampled vorticity. In discrete form, convoluting with the mollifier acts similar to a weighted average, where the coefficients can be chosen from any suitable discrete spatial low-pass filter.
However, in the course of this work it was chosen, that filtering using a mollifier is not optimal for the given framework. The application of the weighted average takes a lot of neighbouring points into account. In order to properly implement it, special care has to be taken for the memory when writing the Cuda kernel function. In addition, the actual filtering effect would have to be properly investigated in order to quantify the behaviour of chosen stencils. Further usage was therefore discarded in regards to other filtering methods.
In the research of this work it was favoured to implement a direct low-pass filter in Fourier space instead. When computing the Biot-Savart-Law, the vorticity is transformed to Fourier-space, where the inverse Laplacian and perpendicular gradient operators are applied. Instead of filtering in real space using the mollification, one can directly filter in the frequency domain. While this features a larger forward Fourier transformation of the vorticity, this computation takes up only a small fraction of the computation of a time-step. In exchange, the actual effect of the filter can be chosen more directly. Currently, a steep low-pass filter is implemented in the framework, where all Fourier modes with a norm $k = \sqrt{k_x^2 + k_y^2}$ larger than the filter strength k_{lp} are set to zero. It is important to note, that due to the entirely real-valued vorticity in real space, the vorticity in Fourier space is symmetric around the Nyquist frequency. The low-pass filter was therefore applied with the same symmetry in both x- and y-direction.

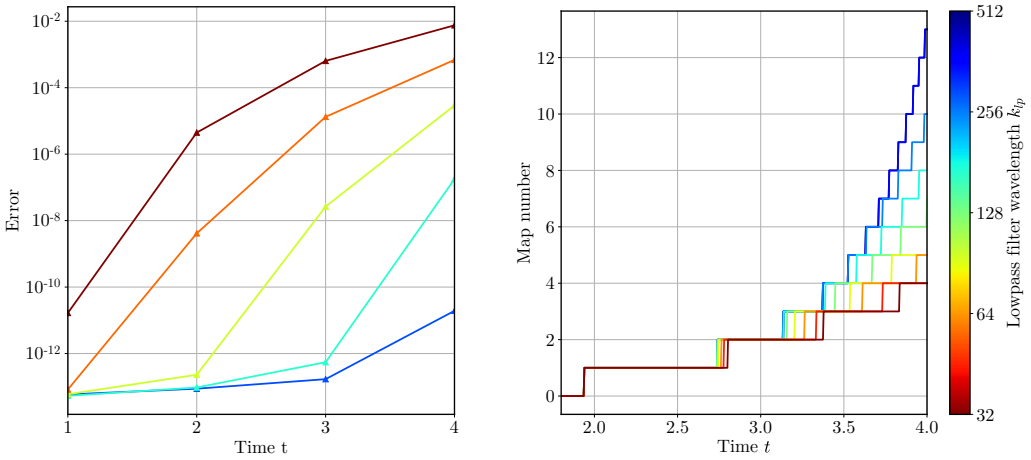

Figure 20: Map error in L_∞-norm to unfiltered computation without remapping (left) and amount of sub-maps with $\delta_{inc,b} = 10^{-4}$ (right) for different filter strengths k_{lp}.

The graphs in figure 20 show different properties of the applied filter. The computations where done with the reference setup with the 4-mode flow from section (4.3). In the left image, the map error can be observed over time for different filter strengths. Featuring initially only very low frequent modes present in the flow, one can observe how each filter takes effect at further time with more and more enstrophy being shifted to higher frequencies. After time, the deviation between filtered and unfiltered flow increases, as the flow now features much more enstrophy in filtered out wavelengths. However, as a lot of enstrophy is starting to non-physically accumulate at frequencies close to the Nyquist-frequency, this error in the flow map does not at all have to be non-physical.
In the right image the large benefits of the application of the filter is depicted. There, the amount of occurred remappings is plotted over the time for different filter strengths. The filter greatly reduces the exponential growth of needed sub-maps. In fact, the amount of remappings per time approaches a maximum when applying a filter, making long time simulations possible.
As the remapping step is very computationally costly and the amount of sub-maps is limited to available memory, typically an optimal filter strength has to be chosen. In larger simulations it was observed, that the applied filter strength plays a major part for the computational cost of a simulation. In future work, the filter strength could be automatically adapted to only filter out the non-physical accumulation of enstrophy in high frequencies. The needed details can be extracted from an enstrophy power-spectrum.

5 Validation of implementation of fluid and inertial particles

While the implementation of point particles being advected by the velocity of the flow was already introduced by Nicolas Saber in [15], the implementation of the time-stepping schemes itself was not yet validated. In this section, all the different schemes from first to fourth order are derived to be able to arbitrarily choose the convergence order similar to the advection of the flow map. Afterwards, the implementation is validated with simulations aiming to show the actual convergence and the performance is discussed.

5.1 Equations of motion for fluid and inertial particles

Currently, there are two different types of advected point particles implemented in the framework. The first is a rather simple approach and computes the advection of fluid particles by the fluid velocity. For that case, the velocity of the particles $\vec{u}_p$ is equal to the velocity of the flow $\vec{u}_f$ at the particle position $\vec{x}_p$:

$$\frac{d}{dt}\vec{x}_p = \vec{u}_p = \vec{u}_f \tag{5.1}$$

This approach should yield equal results to fluid particles being transported via the forward map χ^+.
The other types of particles are inertial particles. Those particles have a velocity on their own and do not strictly follow the fluid flow, but are rather dragged along and accelerated by the surrounding medium. In order to model this drag, the equations of motion by Maxey & Riley in [12] have been used.

$$\frac{d}{dt}\vec{x}_p(t) = F_x(\vec{x}_p(t), \vec{u}_p(t), t) = \vec{u}_p(t) \tag{5.2}$$

$$\frac{d}{dt}\vec{u}_p(t) = F_u(\vec{x}_p(t), \vec{u}_p(t), t) = -\frac{\vec{u}_p(t) - \vec{u}_f(\vec{x}_p(t), t)}{\tau_p} \tag{5.3}$$

The definition of the functions F_x and F_u is used to show the coupling of those two ordinary differential equations and are handy to derive the time-stepping schemes. The factor τ_p as the particle relaxation time depicts how resistant a particle is against acceleration and deceleration due to the drag forces acting on the particle. Further elaboration on that can be taken from [15].
The initial velocity for inertial particles is set to zero, but can chosen to be set after the fluid velocity as well in case stability issues arise. The initial positions of the particles can be chosen arbitrarily.

5.2 Different time-stepping schemes for advection of fluid particles

All implementations of time-stepping schemes used are Runge-Kutta methods of different convergence order and are similar to the schemes for the advected flow maps.

Only the modified third and fourth order methods have been adapted to represent more optimal schemes. Since the advection of the particles is computed after the computation of the stream function in Hermite form, the velocity of the fluid flow is available at times $\{t_{n+1}, ..., t_{n+2-l}\}$, where l is the chosen order of Lagrange interpolation of the velocity values. This is beneficial, as the fluid velocity at time t_{n+1} is needed in all schemes and therefore does not have to be Lagrange extrapolated anymore.
In this section, the detailed equations for inertial particles is shown for the modified third order Runge-Kutta scheme. No further derivations will be given for other implementations of the inertial particles and all implementations for the fluid particles, as the derivation is straight forward from the illustrated case.
By closer inspection of the equations (5.2) and (5.3), the computation for intermediate time-steps k_i of the Runge-Kutta schemes can be optimised to use as little computations as possible with the following equations:

$$\vec{k_{x_i}} = F_x(\vec{x}(t_{n_i}), \vec{u}_p(t_{n_i}), t_{n_i}) = \vec{u}_p(t_{n_i})$$
$$\vec{k_{v_i}} = F_v(\vec{x}(t_{n_i}), \vec{u}_p(t_{n_i}), t_{n_i}) = -\frac{\vec{k_{x_i}}(t_{n_i}) - \vec{u}_f(\vec{x}_p(t_i), t_i)}{\tau}$$

With this, usages of the intermediate velocity $\vec{u}_p(t_i)$ from $\vec{k_{x_i}}$ are reused in the computation of $\vec{k_{v_i}}$. While this seems only like a small change, optimisations like this add up drastically for long computations. This is especially important for GPU computations, as loading and storing values to global memory (here $\vec{x}$, $\vec{u}_p$ and $\vec{u}_f$) is very expensive. In fact, by temporarily saving $\vec{x}_p(t_n)$ in local memory, only one read from global memory has to be done for each the particle position and velocity, as $\vec{u}_p(t_n)$ will always be loaded by $\vec{k_{x_1}}$ for all explicit Runge-Kutta schemes. This results in an optimal memory handling for the particle data. However, the bigger concern is the velocity, where each value has to be interpolated from the Hermite representation of the stream function, resulting in a load of many global memory values for one velocity value. Therefore, a Lagrange interpolation of the velocity, as done in the map advection, should be avoided at all times to minimise the computations and also loads from global memory done.
The special third order Runge-Kutta scheme was chosen to aim to reduce the amount of Lagrange interpolations for the fluid velocity. This constitutes to the general third order case I from ([4] p.186) with $c_2 = -1$ and $c_3 = 1$. Those special time-steps where chosen, as the velocity of the flow is available at all those times. Interpolating the velocity with Lagrange interpolation is therefore not needed, which greatly speeds up the computation.
The Butcher table for this implementation is:

0			
1	1		
-1	1	-2	
	2/3	5/12	-1/12

From this, the general formula with all three intermediate steps becomes:

$$\begin{aligned} k_1 &= F(x_n, t_n) \\ k_2 &= F(x_n + \Delta t/2 \cdot k_1, t_{n+1}) \\ k_3 &= F(x_n + \Delta t \cdot k_1 - 2\Delta t \cdot k_2, t_{n-1}) \\ x_{n+1} &= x_n + \Delta t/12 \cdot (8 \cdot k_1 + 5 \cdot k_2 - k_3) \end{aligned}$$

When applied to the inertial particles, one gets the following equations:

$$\begin{aligned} \vec{k_{x_1}} &= F_x(\vec{x}_p(t_n), \vec{u}_p(t_n), t_n) \\ &= \vec{u}_p(t_n) \\ \vec{k_{v_1}} &= F_u(\vec{x}_p(t_n), \vec{u}_p(t_n), t_n) \\ &= -\frac{1}{\tau_p}(\vec{k_{x_1}} - \vec{u}_f(\vec{x}(t_n), t_n)) \end{aligned}$$

$$\begin{aligned} \vec{k_{x_2}} &= F_x(\vec{x}(t_n) + \Delta t \cdot \vec{k_{x_1}}, \vec{u}_p(t_n) + \Delta t \cdot \vec{k_{v_1}}, t_{n+1}) \\ &= \vec{u}_p(t_n) + \Delta t \cdot \vec{k_{v_1}} \\ \vec{k_{v_2}} &= F_v(\vec{x}(t_n) + \Delta t \cdot \vec{k_{x_1}}, \vec{u}_p(t_n) + \Delta t \cdot \vec{k_{v_1}}, t_{n+1}) \\ &= -\frac{1}{\tau_p}(\vec{k_{x_2}} - \vec{u}_f((\vec{x}(t_n) + \Delta t \cdot \vec{k_{x_1}}), t_{n+1}) \end{aligned}$$

$$\begin{aligned} \vec{k_{x_3}} &= F_x(\vec{x}(t_n) + \Delta t \cdot \vec{k_{x_1}} - 2\Delta t \cdot \vec{k_{x_2}}, \vec{u}_p(t_n) + \Delta t \cdot \vec{k_{v_1}} - 2\Delta t \cdot \vec{k_{v_2}}, t_{n-1}) \\ &= \vec{u}_p(t_n) + \Delta t \cdot \vec{k_{v_1}} - 2\Delta t \cdot \vec{k_{v_2}} \\ \vec{k_{v_3}} &= F_v(\vec{x}(t_n) + \Delta t \cdot \vec{k_{x_1}} - 2\Delta t \cdot \vec{k_{x_2}}, \vec{u}_p(t_n) + \Delta t \cdot \vec{k_{v_1}} - 2\Delta t \cdot \vec{k_{v_2}}, t_{n-1}) \\ &= -\frac{1}{\tau_p}(\vec{k_{x_3}} - \vec{u}_f(\vec{x}(t_n) + \Delta t \cdot \vec{k_{x_1}} - 2\Delta t \cdot \vec{k_{x_2}}), t_{n-1}) \end{aligned}$$

$$\begin{aligned} \vec{x}_p(t_{n+1}) &= \vec{x}_p(t_n) + \frac{\Delta t}{12} \cdot (8 \cdot \vec{k_{x_1}} + 5 \cdot \vec{k_{x_2}} - \vec{k_{x_3}}) \\ \vec{u}_p(t_{n+1}) &= \vec{u}_p(t_n) + \frac{\Delta t}{12} \cdot (8 \cdot \vec{k_{v_1}} + 5 \cdot \vec{k_{v_2}} - \vec{k_{v_3}}) \end{aligned}$$

While this specially chosen method may not achieve as good error bounds as more classical methods with time instants withing the integration interval $[t_n, t_{n+1}]$, the speedup aims to improve the overall error over computational complexity.

5.3 Convergence order for the particle time-stepping schemes

To show the convergence of the methods, all other error factors have to be mitigated and kept constant. However, as the flow of the particles is influenced by the fluid flow

and the computations are dependent on the fluid velocity at specific previous time-steps, the error of the flow map and particle computation are coupled. Due to this, the convergence was computed externally from the main computation. The flow was evolved until a time $T = 1$, afterwards the particles are embedded and computed with stationary fluid velocity ($u_i = u_n$for all $t_i \in [T, T_p]$) until time $T_p = 2$ with different time-step sizes $\Delta t \in \{1/8, 1/16, 1/32, 1/64, 1/128, 1/256, 1/512\}$, similar to the time convergence study of the flow map. The different computations are compared against the computation with $\Delta t = 1/512$ as a reference to compute the particle error:

$$\text{Particle error} = ||P_{ref}(\cdot, T_p) - P(\cdot, T_p)||_\infty$$

The computations are done for fluid particles with $\tau_p = 0$ and inertial particles with $\tau_p = 0.5$. Since the computation of the particles are independent of the fluid flow computation, the parameters can be chosen arbitrarily. The set of parameters being used can be found in the following table:

Name	Value	Name	Value
N_{coarse}	1024	N_{fine}	2048
N_ψ	1024	N_ω	2048
h_{fluid}	1/128		
ϵ_m	10^{-3}	k_{lp}	512
Fluid time scheme	RK3	Map update stencil	4th order
Inital condition	4-mode-flow	Number of particles	10^6

Table 6: Simulation parameter for particle convergence simulations.

Figure 21 shows the convergence behaviour for all available methods for both the fluid and inertial particles. In general, all used methods mainly show the expected convergence behaviour. Also, as expected, the modified third and fourth order methods show a marginally worse performance due to their less accurate schemes chosen.

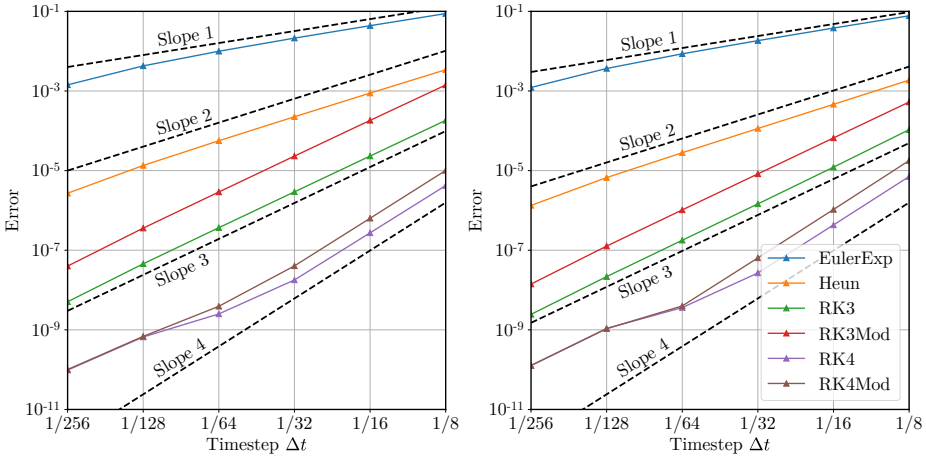

Figure 21: Convergence errors for the fluid particles (left) and inertial particles with $\tau_p = 0.5$ (right) with respective orders in dashed lines.

A saturation effect can be observed starting at an error of around 10^{-8}. This throttles the convergence behaviour for the fourth order methods, whose convergence order is afterwards reduced by one.

5.3.1 Timing for the particle time-stepping schemes

As the previous convergence tests featured a completely decoupled particle loop, the computations of the particle advection itself can be directly timed. The different computations done were used to compute an estimate of deviation in the computed averages for each set. The variance helps to give an estimate on the stability of the measured times and was computed the same way as done in section (4.3.3).
The measurements were taken on a machine featuring an Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz with 8 logical CPUs and 16GB of RAM, paired with a NVIDIA GeForce 845M with 2GB of video memory (VRAM). The timings should aim to give a relative comparison between the methods.
The results can be found in the following table:

Name	EulerExp	Heun	RK3	RK3 Mod	RK4	RK4 Mod
$\tilde{t}_{C,mean}$ [s]	0.1	0.212	0.578	0.342	0.929	0.704
$\sigma_t/\tilde{t}_{C,mean}$	0.001	0.003	0.002	0.004	0.002	0.001
$\tilde{t}_{C,mean,rel}$	1	2.131	5.8	3.437	9.325	7.065
$\tilde{t}_{C,mean}$ [s]	0.11	0.219	0.579	0.344	0.93	0.704
$\sigma_t/\tilde{t}_{C,mean}$	0.005	0.005	0.004	0.003	0.003	0.004
$\tilde{t}_{C,mean,rel}$	1	1.986	5.256	3.118	8.440	6.389

Table 7: Time per step, variance and relative time to EulerExp for fluid (top) and inertial particles (bottom).

Overall, the average variance is about or lower than 0.5%, so that the stochastic difference in computed steps per time between the simulations of one set can be neglected. As no Lagrange interpolation is used for the first order explicit Euler, second order Heun and modified third order Runge-Kutta method, their computational cost scales almost linearly in respect to each other due to the amount of intermediate estimates being computed. This behaviour is supported by the timings. Once a Lagrange interpolation is needed, the time per step is increased drastically, but still scales roughly with the amount of total Hermite interpolations being done. Also, the advection of the fluid particles and inertial particles takes almost exactly the same time, again giving a strong link to the importance of the Hermite interpolation.
In order to further assess the qualities of the methods, the computation time plotted over the achieved error for the fluid and inertial particles can be seen in figure 22. As per definition of the axes, a curve closer to the origin depicts a faster and more accurate method. The first order method is not efficient at all and with each increase in convergence order, the methods become more and more efficient, as the cost only increases linearly. Interestingly, the modified versions of the higher order methods do not achieve largely more efficient results. Especially for the third order methods, the classical Runge-Kutta scheme slightly outperforms the modified version. However, the modified fourth order version does improve the efficiency in comparison to the classical method, while still being almost as fast as the classical third order method. Considering purely the particle methods not embedded in fluid flow, the fourth order modified version proves to be the most efficient one and is recommended to be used for simulations.

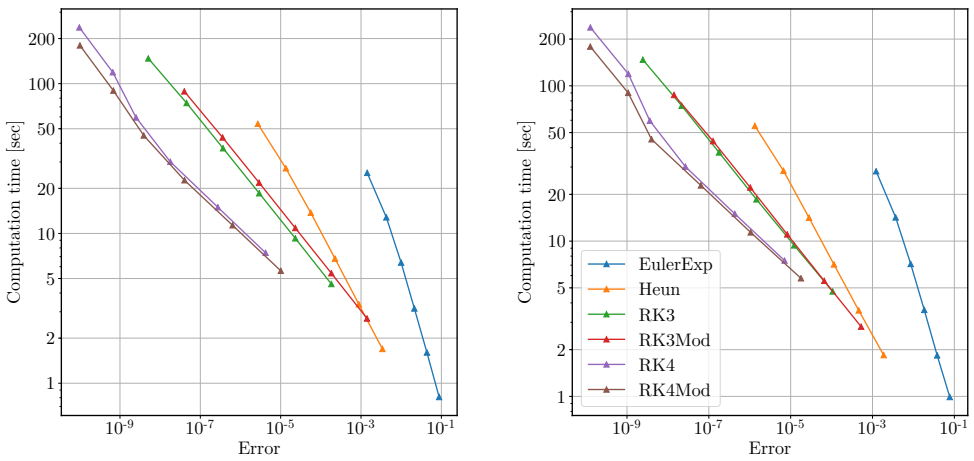

Figure 22: Computational time over achieved particle position error for fluid (left) and inertial particles with $\tau_p = 0.5$ (right) and different time-stepping method.

5.3.2 Convergence order and timing for the particle time-stepping schemes embedded in fluid flow

While the previous tests well showed the convergence behaviour of the different particle time-stepping methods, it did not represent real working conditions. Usually, the particle error for computations with the framework will always be connected to the flow map error. To assess the difference, another convergence test has been conducted, however this time the particles were embedded in the fluid flow right from the beginning and the time-step Δt was therefore changed for the flow map and particle computation simultaneously.

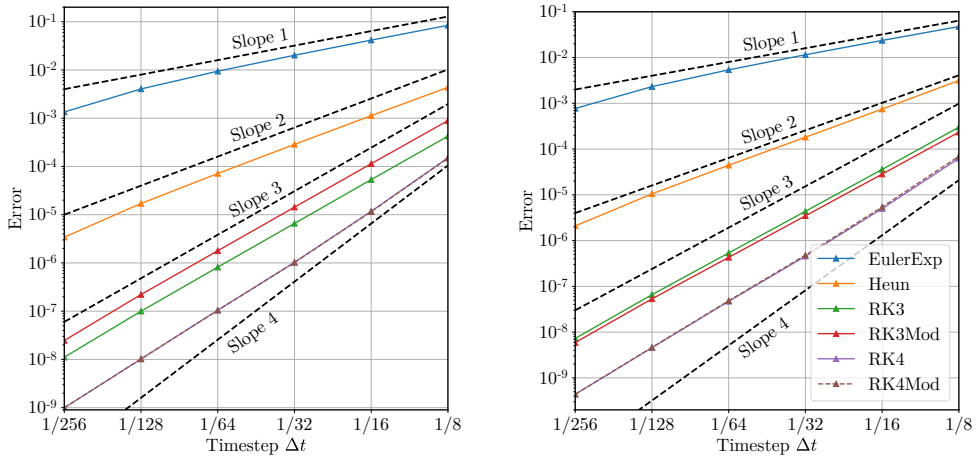

Figure 23: Convergence errors for the fluid (left) and inertial particles with $\tau_p = 0.5$ (right) embedded in fluid flow. The convergence for both 4th order methods is identical.

Surprisingly, the convergence behaviour is not changed much. Mainly, the property of the third and fourth order methods is improved. For the two third order methods, the behaviour is improved and both methods show also smaller deviation. Both fourth order methods coincide with each other and have decreased performance. As the velocity is computed to third order accuracy in time, the decreased performance is actually anticipated. However, these higher order methods surprisingly are still able to compute the particle positions more accurately. Which presumably comes from the increased 4th order of Lagrange interpolation needed for those higher order implementations.

Name	EulerExp	Heun	RK3	RK3 Mod	RK4	RK4 Mod
$\tilde{t}_{C,mean}$ [s]	0.976	1.198	1.914	1.442	3.133	2.466
σ_t	0.005	0.004	0.002	0.005	0.003	0.003
$\tilde{t}_{C,mean,rel}$	1	1.227	1.961	1.478	3.21	2.527

Table 8: Time per step, variance and relative time to EulerExp for combined fluid and inertial particle sets embedded in fluid flow. A third order Runge-Kutta method was used to advect the flow map.

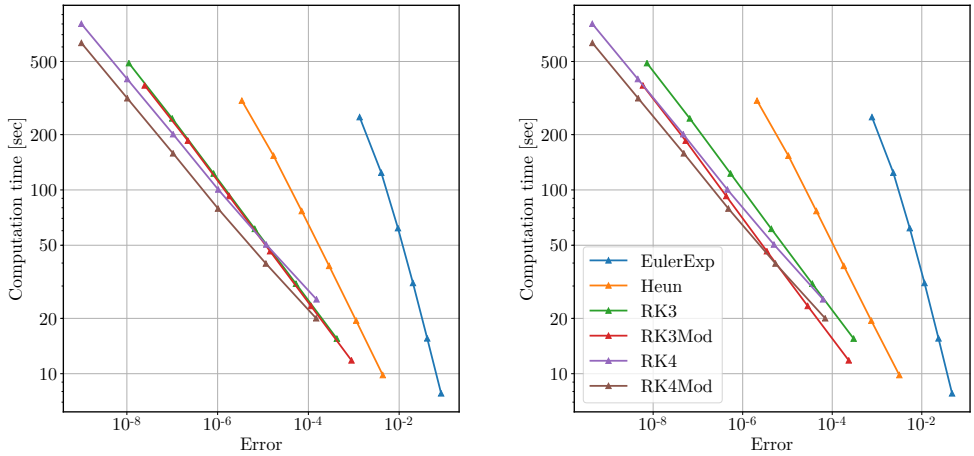

Figure 24: Computational time over achieved particle position error for fluid (left) and inertial particles with $\tau_p = 0.5$ (right) embedded in fluid flow.

Again, the timing for each computation was measured in order to assess the efficiency of each method. Due to the advecting of fluid and inertial particles being almost identically fast, they have been computed together at the same time. In table 8, the time per step, variance and relative time are shown similar to the preceding section. As the number of advected particles with 10^6 is almost equal to the number of points inside the flow map with 1024^2, the computation length for the particle computation starts to dominate the simulation from those times, which reach an increase of two or more times in comparison to computations with the first order method. Especially the fourth order methods take more time due to the increased order of Lagrange interpolation.
However, as can be seen in figure 23, the increased computational cost for higher order methods is made up for in terms of accuracy, yielding more efficient results. Both modified versions are more efficient than their classical counterpart, with the modified third order variant even exceeding the classical fourth order method for inertial particles. When enough time and resources on hand, the modified fourth order method gives excellent results while still being sufficiently fast. For fully third order methods, the third order modified version exceeds its classical counterpart and is to be preferred, especially when implementing small time-steps.

6 Scalar mixing in shear layer flow

Besides the movement of the vorticity or velocity, investigating the flow of other passive phenomena not directly influencing the fluid flow is of high interest too. This has many application fields, for example the investigation of several different fluids or concentration fields. Those can be used to study chemical reaction rates or mixing properties.
As turbulence is an important phenomena to support mixing of different quantities, there is a particular interest in investigating the fine structures of mixing fluids due to turbulent flow.
In this chapter, a test-case will be setup and used to study the mixing behaviour of two fluids with opposing velocity, shearing against each other. Kelvin-Helmholtz instabilities arise and the two fluids start to mix more and more at the boundary zone. This mixing is qualitatively investigated in order to show-case the effectiveness of the magnifying properties of the characteristic mapping method.

6.1 Scalar transport with the characteristic mapping method

The characteristic mapping method does not only solve the transport equation for the vorticity, but can solve it for any passive scalar quantity θ being advected by the velocity $\vec{u}$:

$$\frac{d\theta}{dt} = \partial_t \theta + \vec{u} \cdot \nabla \theta = 0 \tag{6.1}$$

$$\theta(\vec{x}, 0) = \theta_0(\vec{x}) \tag{6.2}$$

By definition, any passive scalar is advected by use of the characteristic flow map χ. Similar to the vorticity, the solution at time t can be traced back using the backwards flow map:

$$\theta(\vec{x}, t) = \theta_0(\chi^-(\vec{x}, t)) \tag{6.3}$$

Therefore, solving a passive scalar can also benefit from the semi-group structure. As the backwards flow map is already computed, computing the advection of passive scalars is imposing very little additional computational cost.

6.2 Initial condition - shear layer flow

To investigate the mixing behaviour, a typical reference flow was chosen. This features a shear flow of two fluids with opposing velocity direction. A thin transition layer between the two fluids creates a vorticity profile being present at the boundary zone and zero everywhere else. Physically, the flow is inviscidly unstable and perturbation leads to the formation of Kelvin-Helmholtz instabilities in the form of multiple vortices of equal direction. In order to accelerate the emergence of this phenomena numerically, an instability of the same frequency was superimposed. An

explanation of this problem can be found for example in [16].
The initial velocity profile features a boundary layer with a hyperbolic tangent profile. When applying the curl operator to get the vorticity, it becomes

$$\omega_0(x, y) = c_1(1 + c_2 \cos(c_3 x)) \cdot \operatorname{sech}^2(-c_4(y - \pi)) \tag{6.4}$$

Here, sech is the hyperbolic secant. The factors have different influence, which can be extracted from table 9. While c_1 to c_3 can be set freely, the thickness of the boundary profile c_4 is physically connected to the most amplified mode.

Parameter	Definition	Value
c_1	Strength of mean stream velocity	5
c_2	Strength of perturbation	0.01
c_3	Frequency of perturbation	2
c_4	Thickness of boundary profile	$\frac{14c_3}{\pi}$

Table 9: Different parameters of the shear flow initial condition with hyperbolic tangent velocity profile.

The strength of the mean stream velocity defines how fast the flow develops and merges. The frequency of the instability corresponds to the amount of Kelvin-Helmholtz instabilities, which are formed. Usually the boundary thickness physically defines the amount of vortices, but it was chosen the other way round in order to preset these. While a shear flow initial condition was already presented in the work of [15], it did not feature a hyperbolic tangent profile for the velocity as well as the physical distinction of the parameters.
The simulations to achieve the result featured the highest N_{coarse} available for the CPU RAM of 190GB on the machine running them. The set of parameters used were:

Name	Value	Name	Value
N_{coarse}	2048	N_{fine}	8192
N_ψ	4096	N_ω	8192
h_{fluid}	1/6144	$\delta_{inc,b}$	$5 \cdot 10^{-4}$
ϵ_m	10^{-3}	k_{LP}	256
Fluid time scheme	RK3	Map update stencil	4th order

Table 10: Simulation parameters for the shear layer flow.

The low-pass filter strength and incompressibility threshold were chosen to balance the amount of remappings. All variables for investigations where captured on a 4096^2 grid. As depicted in figure 25, the initial vorticity profile rolls up in clock-wise direction to form two Kelvin-Helmholtz vortices, one at the center of the computational domain and one at the periodic boundary. Interaction between the outer layers of the two vortices form very thin vorticity structures. Eventually, at a time

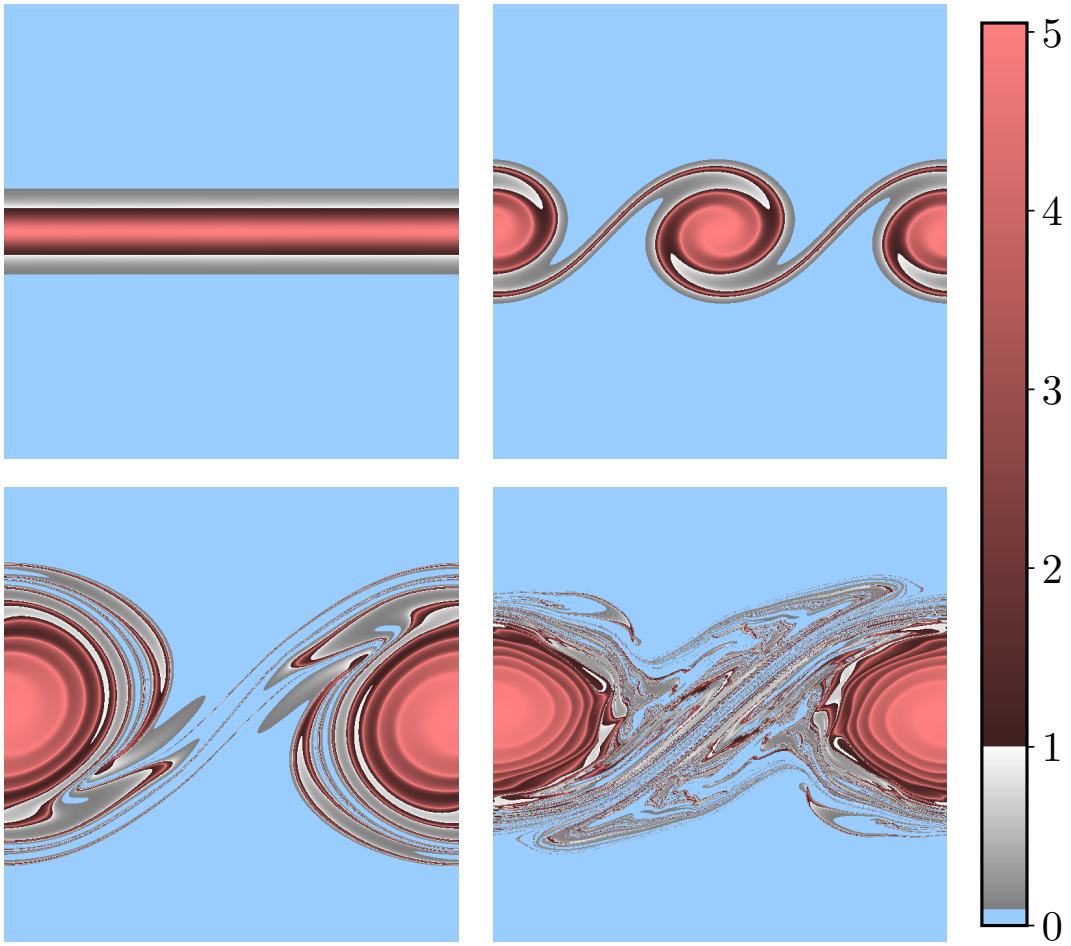

Figure 25: Vorticity profile for the shear flow simulation at times $t = 0$ (upper left), $t = 8$ (upper right) over the whole domain and zooms between the vortices at times $t = 15$ (lower left) and $t = 25$ (lower right).

of around $t = 30$, the symmetry between the two vortices is broken and they start to merge.

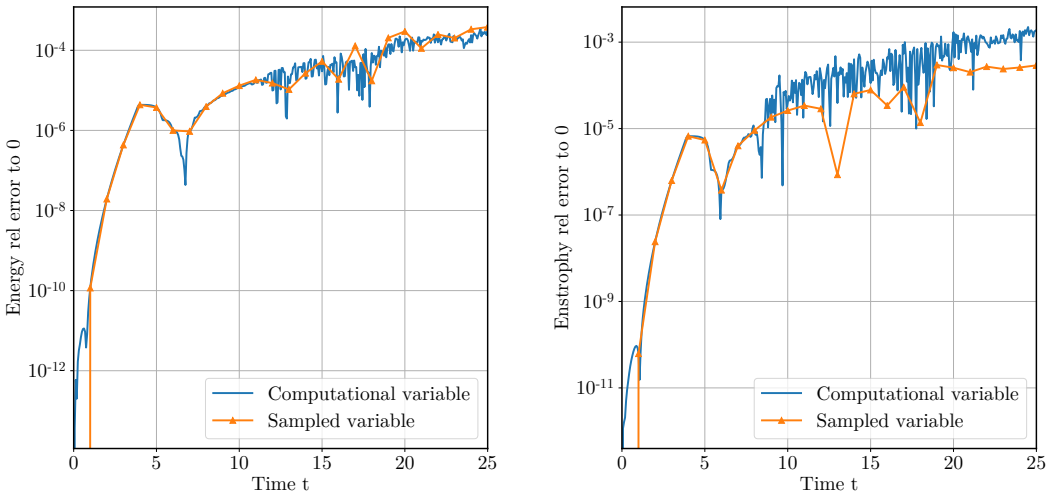

Figure 26: Relative conservation of energy (left) and enstrophy (right) to time $t = 0$: $(E(t) - E(t = 0))/E(t = 0)$.

The conservation properties of the simulations are shown in figure 26. The values for the computational variable where taken from the individual vorticity and stream function used inside the computation and where taken at a high rate, while the values for the sampled variable where computed after application of all individual sub-maps at that time instant. As all saved variables are sampled, their conservation corresponds to the conservation of the result, while the computational variables provide information on the stability of the computation.
After an initial phase where the two individual vortices are formed, the conservation of the computational variable starts to oscillate, hinting on stronger partial deformation of the maps. However, overall the error growth is bounded and stays at small magnitudes below 10^{-3}.

6.3 Mixing in shear layer flow

An important property of turbulence is the increased amount of mixing and transport occurring. With the widespread presence of shear layer flow in applications and its clear developing coherent structures, it makes an excellent test-case to qualitatively study the mixing behaviour of scalar quantities in turbulent flow.
The following observations were made comparatively to [3], where the transport in shear layer flow was investigated using a pseudo-spectral method. Using the CM-method allows to greatly focus on the fine scale structures emerging in comparison to earlier work.
In order to differentiate between the two fluids which are shearing alongside each other, a scalar with value $\theta = 1$ for the lower, and $\theta = 0$ for the upper half plane was embedded into the flow. Due to the backwards map always pointing to a discrete location and the initial condition being analytical, the scalar θ will always have dis-

crete values of zero or one, no matter the quality of the flow map. This also means, that flow transported by the incompressible Euler equations does not actually mix, but only further and further intertwines the two fluids. In addition, the transport equation implies that the scalar quantity has to be conserved in the scope of the simulation. In fact, the achieved conservation was quite stable over the course of the simulation, with a maximum relative conservation error of $7.34 \cdot 10^{-4}$ in comparison to the sum of the initial scalar.

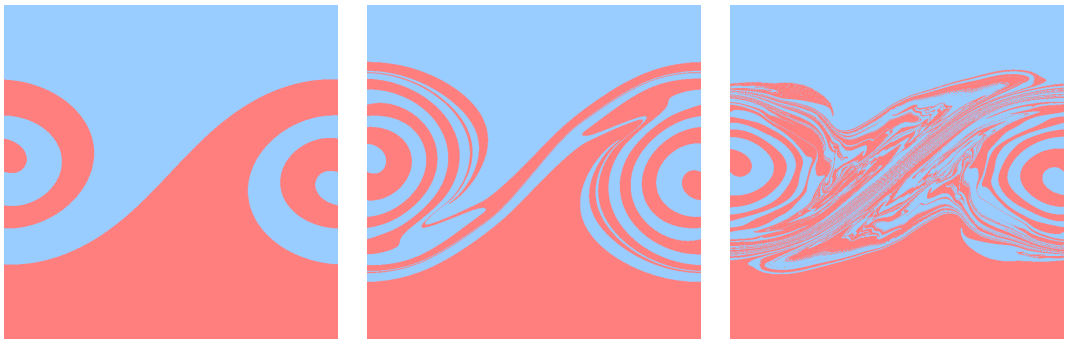

Figure 27: Zoom of passive scalar between the two vortices at times $t = 8$ (left), $t = 15$ (middle) and $t = 25$ (right).

Figure 27 shows the behaviour of the passive scalar over time. The formation of the vortices forms swirls of the two sides in its center. In addition, the two vortices interact and scalar parts that are sucked into the other domain will later travel to the other vortex. As an example, the red half-circle visible in the left image will continue to move to the right, eventually forming the thin sheet visible at the center image. This leads to the formation of a very thin folded structure, which is visible at later times.
Due to the zoom-property of the method, the fine structures can be further analysed, as all of them are preserved within the composition of the sub-maps. Enlarging the area at the center between the vortices shows, how the two scalars are continuously folded. The left image in figure 28 shows a close-up at time $t = 15$. The image represents the square with side-length $\pi/4$. The firstly explained folding is here clearly visible. A close-up at later times captures the highly-folded structure. While the outer parts start to get irregular, this structure still follows the original division line visible in the left image of figure 27. Further zooming in to a frame with side-length $\pi/16$ visualises the thin sheet-like structures. While initially barely visible in the right image of figure 27, all of the fine structures are still preserved within the full map-stack itself. This feature proves to be a big advantage of the method, as capturing fine structures is not dependent on the initial grid. However, due to the incompressibility error of the map, artificial structures can arise as well, which can quickly reach sizes of those fine structures investigated here.
Together with the folding structure comes a large interface between the two scalar

parts. This is highly beneficial for dissipation processes, resulting in blending of the two individual parts. In addition, reaction processes benefit from large interfaces as well, further accelerating the rate of reactions happening.

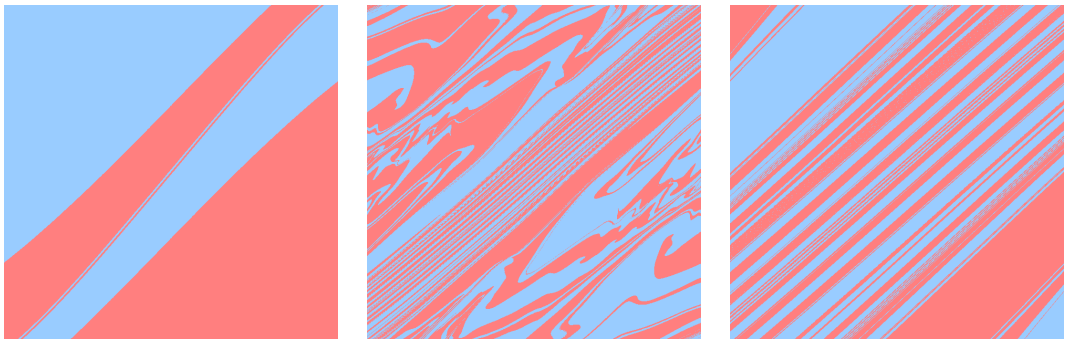

Figure 28: 8-folded zoom of the passive scalar between the two vortices at times $t = 15$ (left) and $t = 25$ (middle) and 32-folded zoom at $t = 25$ (right).

7 Particle transport in fully developed turbulence

Particles suspended in turbulent flow is a common real-world scenario. From impurities in atmospheric flow acting as seeds for cloud formation, pollution and virus spreading inside cities to bacteria in liquid flows - aerosols of particles and fluids is an important occurring phenomena, especially in the field of geophysical flow. This is for example largely argued by Matsuda et al. in [11], where they investigated the statistics of inertial particles in turbulent flow. Especially the reaction of particles to turbulent flow is a current, interesting topic, due to the advancements made in computing turbulent fields. A special case of turbulence, called isotropic homogeneous turbulence, gives ideal condition in order to investigate the influence of turbulent flow on particles. A turbulent fluid is isotropic and homogeneous, if it is statistically invariant to translations and rotations. This means, that all statistical quantities are everywhere and in all directions equal. Effectively, small portions of an isotropic homogeneous turbulent domain are again isotropic and homogeneous. This gives some kind of self-resemblance, comparable to fractal dynamics. In real world, this can be created by for example running a fine mesh through a domain, creating turbulence along its path without imposing a mean flow. Numerically however, often real world data has to be used in order to impose this kind of turbulent flow. Another option is to create turbulence with a suitable instable initial condition. In order to have an analytical initial condition, the latter was chosen in this work to create a state of isotropic homogeneous turbulence.

7.1 Initial condition for 2D homogeneous isotropic turbulence

In order to generate 2D isotropic turbulence, an unstable initial condition with high enstrophy present in large wavelengths can be constructed. However, it is important that this enstrophy is spread out evenly in order to be as homogeneous as possible. Randomised instabilities lead to enstrophy moving to finer scales due to the enstrophy cascade, eventually forming a state fitting for isotropic homogeneous turbulence. In this work, the approach of Clercx et al. from [8] was adapted. The main idea is to evenly spread out Gaussian vortices with alternating sign in a checker-board pattern. The position of the vortices is randomly perturbed in order to destabilise the flow. In the setup for the following simulations, a field of 10×10 vortices was chosen. Distributed evenly over the domain with size $[0, 2\pi]^2$, each vortex inhibits a space of $d_v = \pi/5$. A radius of expansions of $\sigma_v = d_v/4 = \pi/20$ was chosen, so that each vortex fills out the given space without spreading too much into neighbouring cells. Without perturbation, the vorticity should be zero at the interface between two vortices, with overlapping effects cancelling each other out. For the random perturbation of the vortex positions, a maximum perturbation of $p_v = \sigma_v/2 = \pi/40$ was chosen per dimension. The initial vorticity can then be constructed from superimposing the influence of all vortices. The final equation can be found in equation (7.1).

$$\omega_0(\vec{x}) = \sum_{i_x=1}^{10}\sum_{i_y=1}^{10}\left[\frac{(-1)^{i_x+i_y}}{2\pi\sigma_v^2}\exp\left(\frac{-(\vec{x}-\vec{P}_v(i_x,i_y))^2}{2\sigma^2}\right)\right] \tag{7.1}$$

$$\vec{P}_v(i_x,i_y) = [(i_x+0.5)\frac{\pi}{5} + r_x(i_x), (i_y+0.5)\frac{\pi}{5} + r_y(i_y)]^T \tag{7.2}$$

The position of each vortex is given by the vector $\vec{P}_v(i_x, i_y)$, where $r_x(i_x)$ and $r_y(i_y)$ are the random perturbations for each vortex. The positions are shifted by $d_v/2$ in order to correlate the vortex box boundaries with the domain boundaries. In order to achieve periodicity, the influence of the vortices from the next domains have to be taken into account too. This is done by projecting the border vortices into the next domain and adding their influence too. As the Gaussian vortices scale exponentially, a border size of 2 at each side was chosen to be sufficient. This means, that for each point the influence of 14×14 vortices is added up together. With this approach, the global sum of the vorticity is close to machine precision, as all the vortices cancel each other out in respect to the whole domain.
The simulations where done with largely similar parameters as the ones for the shear layer flow. All variables for investigations where captured on a 4096^2 grid.

Name	Value	Name	Value
N_{coarse}	2048	N_{fine}	8192
N_ψ	4096	N_ω	8192
h_{fluid}	1/6144	$\delta_{inc,b}$	10^{-3}
ϵ_m	10^{-3}	k_{LP}	256
Fluid time scheme	RK3	Map update stencil	4th order

Table 11: Simulation parameters for the isotropic homogeneous turbulence simulations.

The vorticity over time is depicted in the images in figure 29. The slight random displacement is visible in the initial condition, which quickly leads to vortices of the same sign being attracted to each other. This procedure continues and more and more vortices merge. At the same time, the enstrophy cascade shifts more and more enstrophy into smaller scales. As the grid is not able to capture all frequencies at some point, noise appears in later time-steps. The conservation of energy and enstrophy is depicted in figure 30. Similar to the simulation of the shear layer flow, both the conservation for the computational and sampled variables are shown. Noise starts to appear starting at $t = 5$, being especially present in the energy. As the conservation of energy for the computational variable is already low-pass filtered, it also shows, that no visible amounts of the global energy are discarded due to the filtering process. For the enstrophy, the computational variable is not able to capture the enstrophy well. This is due to it being computed on a 2048^2 grid in comparison to the 8192^2 grid of the fine vorticity as an initial condition for each sub-map. This

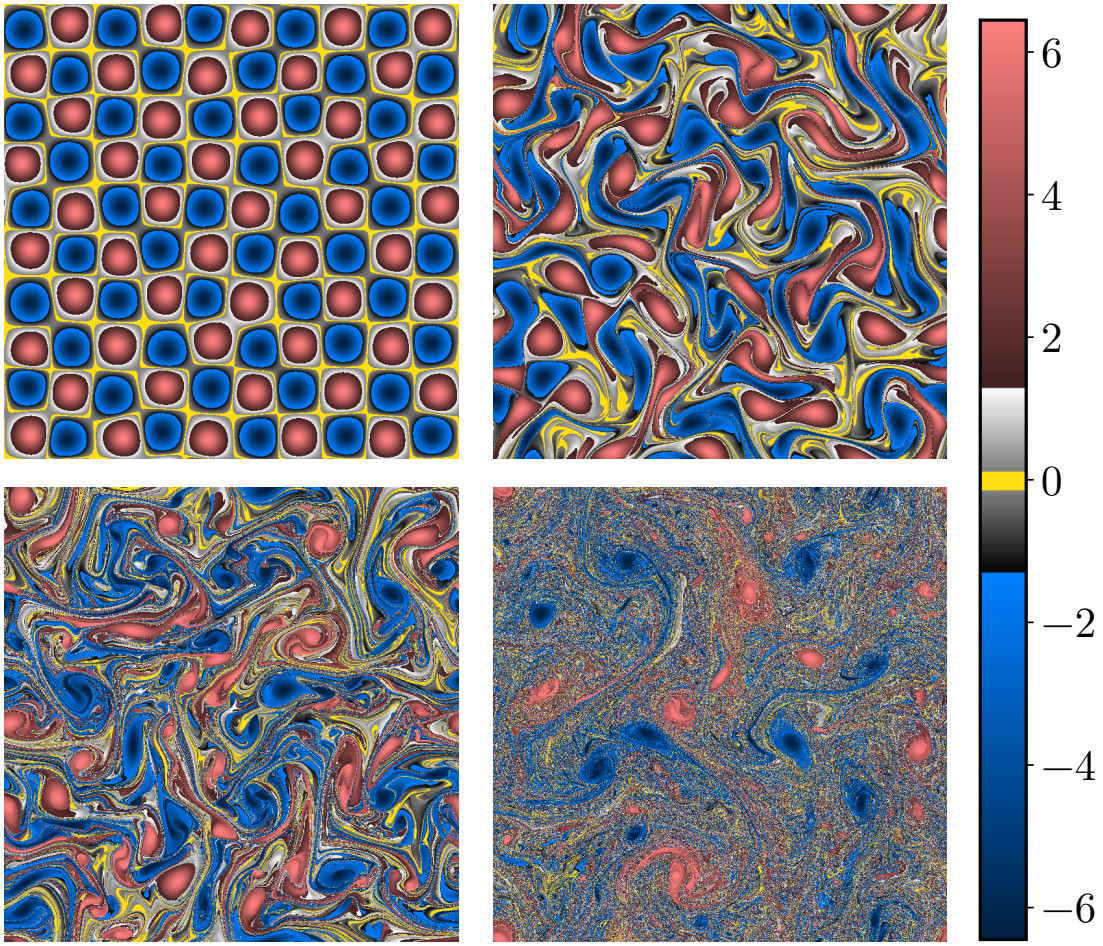

Figure 29: Vorticity for the simulation of isotropic turbulence at times $t = 0$ (upper left), $t = 5$ (upper right), $t = 15$ (lower left) and $t = 25$ (lower right) over the whole domain.

down-sampling discards enstrophy in finer scales and leads to reduced conservation properties. However, those effects are not present for sampled variables.
The palinstrophy is supposed to grow exponentially over time. However, due to the finite grids used to capture vorticity gradients and low-pass filtering, it eventually stagnates and reaches a maximum. Again here the computational and sampled variable differ, with the sampled variable being able to capture more vorticity gradients due to its larger grid-size.

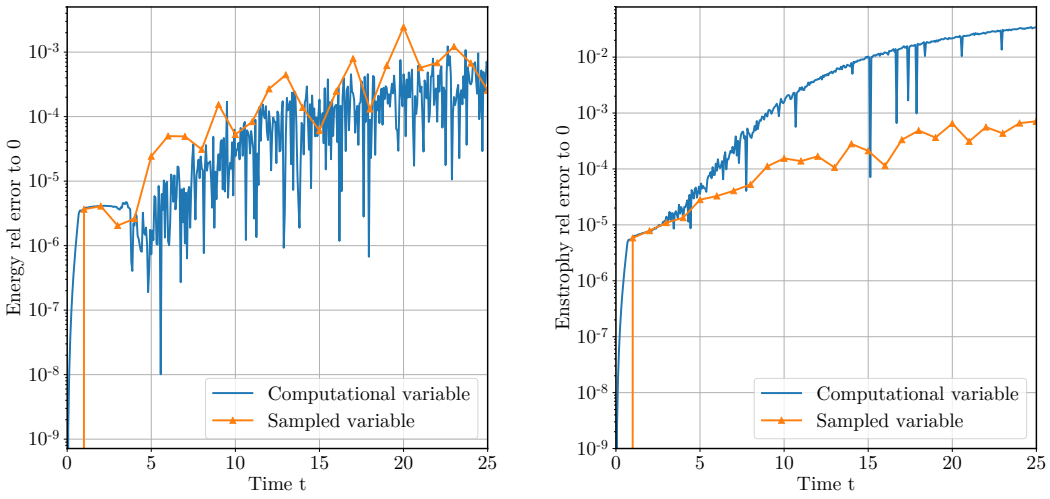

Figure 30: Relative conservation of energy (left) and enstrophy (right) to time $t = 0$.

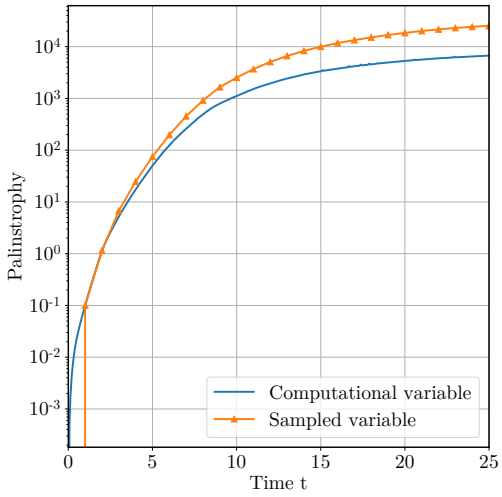

Figure 31: Palinstrophy over time for the simulation of isotropic turbulence.

7.2 Estimation of the time interval suitable for investigations of physical turbulence

Turbulence is a physical behaviour being present only in three-dimensional systems, due to the vortex stretching terms reducing to zero for two-dimensional flow. Due to that, simulations containing real physical turbulence are not possible in the 2D case. However, considering the physically expected behaviour, an interval of homogeneous isotropic turbulence can be defined in planar flow too, which can evolve from a suiting initial condition. A strong condition is the cascade of enstrophy from larger to finer scales and for energy the other way round respectively. While for

the three-dimensional case this follows the widely known Kolmogorov-law with a power-law of exponent $-5/3$ for the relevant scales of the energy, this changes to -3 for the energy and -1 for the enstrophy for two-dimensional flow ([10] p183). This power law has to be validated with the constructed initial condition in order to assure the self-similarity, which is important for turbulent flow.

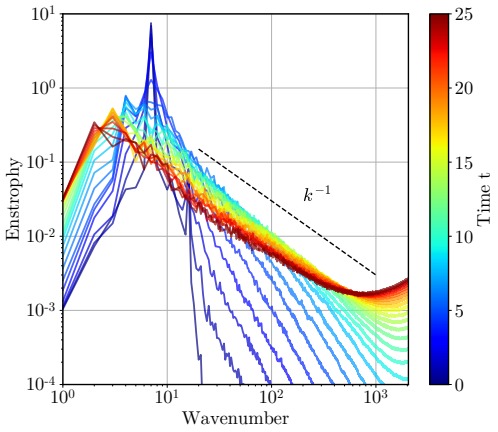

Figure 32: Isotropic spectrum of enstrophy from the vorticity at different times.

Depicted in figure 32 is the isotropic spectrum of the enstrophy for different times until a final time of $t = 25$. The initial spectrum at $t = 0$ shows a strong peak at wavelength $k = 7 \approx 5\sqrt{2}$, coming from the initial checker-board grid of distributed vortex-pairs with 5 pairs per length. With increasing time more and more enstrophy is transported onto finer scales and after some time it starts to pile up for large frequencies around the Nyquist-frequency of the grid being $k_{Nyq} = 2048$. From this transport a power-law for the decay of enstrophy to finer scales is quickly visible, which decreases over time. Also, as more and more vortices merge the peak in enstrophy increases to smaller wave-numbers. Overall, a linear fit can approximate the power-law fairly well. A depiction of that for three different time instants can be found in the left image of figure 33. They were fitted on a range between the initial peak for low wave-numbers and pile-up of enstrophy at high wave-numbers. As many stochastic variations are present and the up-rise in enstrophy at larger wave-numbers for later time instants bends the curve slightly, this deviates it from a definite power-law. Nevertheless, the power-law generally follows a pattern over time with increasing values from -2.75 at the beginning to -0.8 at the end. It reaches the desired power-law for self-similar isotropic turbulence at $t \approx 15$ and stays in a 25%-range starting from $t = 11$. For higher time instants the power-law seems to converge.
For the scope of further investigations it was concluded, that starting from $t = 10$ the generated flow can be assumed to sufficiently approximate physical isotropic

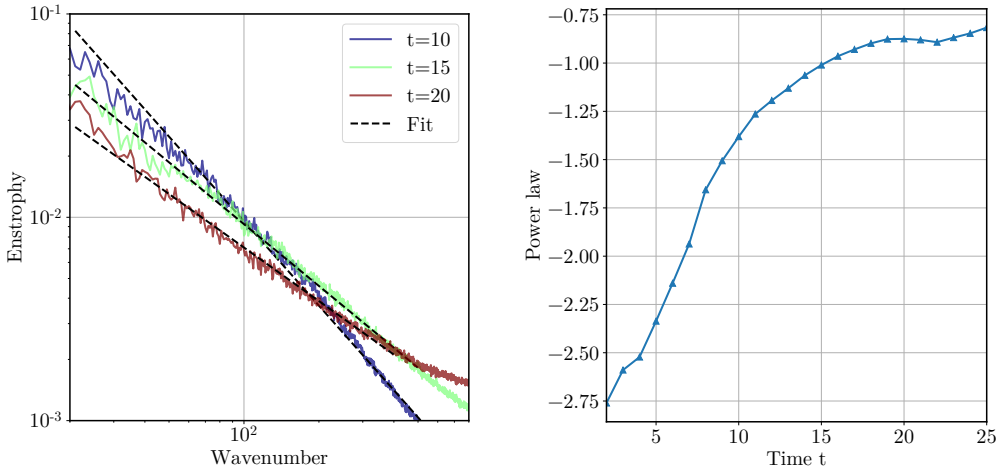

Figure 33: Fitted power-law curves for enstrophy spectra at different time instants (left) and estimated power law over time (right).

homogeneous turbulence.

7.3 Stokes number for embedded inertial particles

Dependent on the inertia of the particles, the characteristics of their behaviour embedded in fluid flow differs greatly. This can be well characterised by the so-called Stokes-number, which according to [15] is defined as

$$St = \frac{\tau_p}{\tau_\eta} \tag{7.3}$$

This number describes the relation between the particle relaxation time τ_p and the Kolmogorov time scale τ_η which can be expressed as

$$\tau_\eta = \left(\frac{\nu}{\epsilon}\right)^{1/2} \tag{7.4}$$

being the relation between the kinematic viscosity ν and the energy dissipation ϵ. Both time scales can be seen as the respecting non-dimensional factors representing the speed in which the velocity reacts to forces. For Stokes numbers $St \ll 1$, the fluid flow defines the trajectory of particles. They are expected to largely follow the fluid stream lines with $St = 0$ representing full fluid particles. With $St \approx 1$, both time scales even itself out and the rate at which the particles reacts to change in fluid flow equals to the change in the fluid velocity itself. For the special case $St = 1$, the particles are expected to cluster in local minima of the norm of the vorticity, which for isotropic turbulence are lines separating clusters of vortices with identical direction. With increasingly large Stokes number

$St \gg 1$, the particle becomes more and more resistant to fluid motion until for $St \to \infty$ it eventually is completely unaffected by the fluid it is immersed in [11]. As for the incompressible Euler equations no viscosity and energy dissipation is present, a Kolmogorov time scale cannot be defined. However, due to the nature of the isotropic turbulence arising from the initial condition, individual particle relaxation times τ_p lead to particle behaviour corresponding to different Stokes numbers. By comparing this to results from other simulations, the Stokes number can be largely estimated. For reference, the work in [11] was used. The particles were uniformly random distributed over the whole domain at time $T = 10$, after which they start to cluster in or avoid specific regions. By comparing the particle density spectrum, a comparison can be made to the reference. Especially the value τ_p leading to $St = 1$ is of interest in order to categorise the particle behaviour into regions of Stokes number larger or smaller than one.

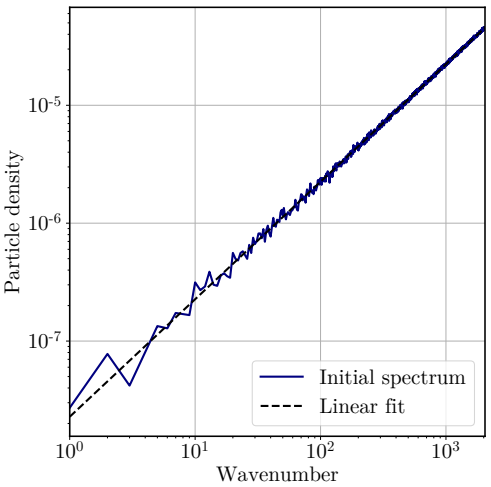

Figure 34: Initial particle spectrum at time $T = 10$.

The particle density was computed by counting the amount of particles in discrete boxes with 4096×4096 uniform boxes distributed over the whole domain. This size of the density grid was chosen in order to coincide with the grid-size for the fluid velocity. In total, eight different sets of particles were embedded at time $T = 10$ with one million particles each. The positions of the particles were computed until a final time of $T = 25$. The positions were saved in an interval of $\Delta T = 0.5$ in order to show the development of the particle spectra over time. Similar to [11], the influence of the initial particle spectrum can be subtracted. As all particle sets shared the same random seed, the initial position and spectrum are identical. With uniformly distributed particles the initial spectrum should experience a power-law with exponent 1, which can be seen in figure 34. A linear fit estimates a power-law of 0.997 with an initial density of $2.28 \cdot 10^{-8}$ at wave-length $k = 1$. For further investigations, the linear fit of the initial spectrum was subtracted. This is done in log-log-scale by division with the initial density for each wavelength.

All particle spectra experience an increase in density with maxima at around $k = 4$ and after some time, the spectra approach stationary states.

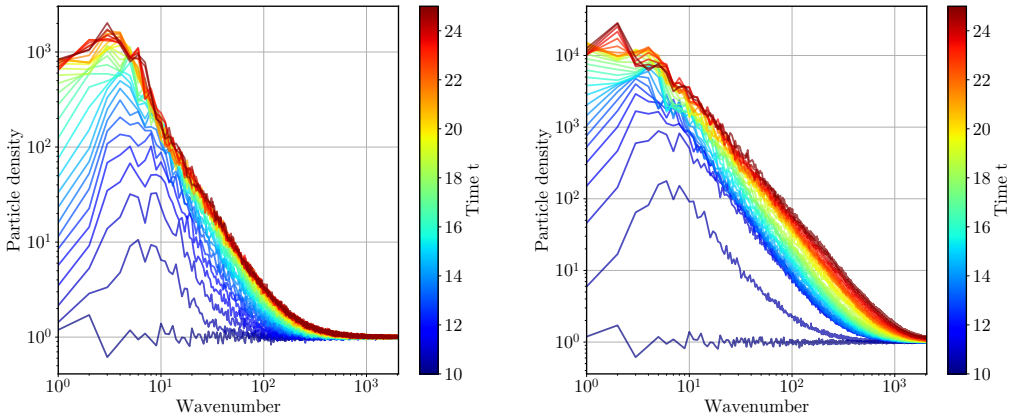

Figure 35: Particle spectrum with $\tau_p = 0.05$ (left) and $\tau_p = 1$ (right) over time.

Two particle spectra over time are visualised in figure 35. The left one depicts a state with low Stokes number, where a stationary state is reached rather quickly. In addition, the maximum particle density is comparably small. The spectra on the right shows a state with larger Stokes number closer to $St = 1$. Not only is the maxima shifting further to $k = 1$, but also is a stationary state is reached slower in the interval of the simulation. Due to the relaxation effects of the fluid and particles evening itself out for $St \approx 1$, the initial condition for the particles needs more time to adapt to the turbulent fluid flow, as more light particles more easily follow the fluid flow and more heavy particles more easily resist it.

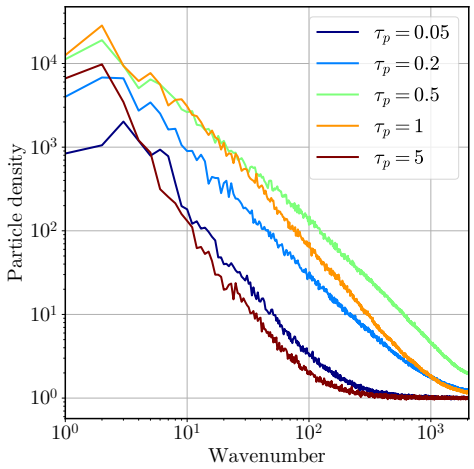

Figure 36: Particle spectrum for different τ_p at time $T = 25$.

With increasing particle relaxation times τ_p, this maxima is further and further

increased until around $\tau_p = 0.5$, at which it reaches a maximum. Afterwards, it starts to decrease again and shifts toward larger wavenumbers. According to [11], a particle distribution with $St = 1$ is reached with the highest maximum, which in the tested case corresponds to $\tau_p \approx 0.5$. Figure 36 depicts the final spectra for different values of τ_p. While the spectra for $\tau_p = 0.5$ and $\tau_p = 1$ show a largely similar maxima, the spectra for $\tau_p = 0.5$ shows higher particle density for larger wavenumbers.

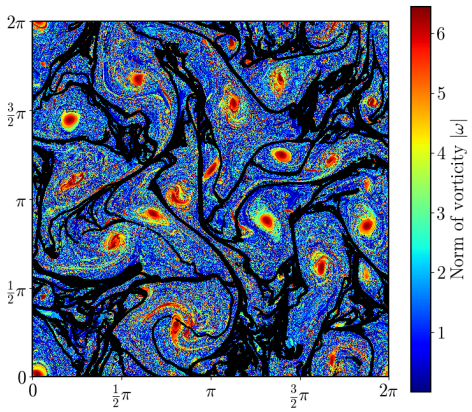

Figure 37: Absolute value of vorticity and particle distribution with $\tau_p = 0.5$ at time $T = 25$.

Figure 37 shows, how the particles with $St \approx 1$ avoid regions with high absolute vorticity. Especially the vortex structures are largely avoided, eventually partitioning the domain into regions containing one vortex each. The regions with more spread out particles all correspond to regions of low absolute vorticity. Considering the spectra and particle distribution, particles with a particle relaxation time $\tau_p = 0.5$ are considered to have Stokes number $St \approx 1$.

7.4 Length of a finite line in turbulent flow

In 1999, Carmona et al. studied the movement of drifters in fluid flow driven by random external forcing in [5]. In their work, they inserted a line of particles oriented in a circle and advected them, to later study the length of the line and how it spreads over the domain. Already in [6] they concluded, that the length of the line growths exponentially in time for isotropic turbulent flows besides shear flow and related that to the positivity of the Lyaponov exponents, which describes the rate of separation of infinitesimally close trajectories.
The simulations with the particle ring in [5] showed strong growth over time and started filling up the plane. From this, they tried to compute the fractal dimension of the particle set, precisely the Minkowski dimension. This is defined as the rate of growth of area, when surrounding the particle set with a tube of size ϵ_M and

changing the tube size. However, the results did not give clear results.
Bec investigated the fractal clustering behaviour in random flows in [2]. He also investigated the Lyaponov exponents and concluded, that the particles are either clustering or space filling dependent on a critical Stokes number. Looking at his definition of the Stokes number it is roughly in proportion to $\frac{1}{\tau_p}$ from the given definition in this work. He defines a critical Stokes number St_c dependent on several factors, which separates the behaviour of the particles. For a Stokes number $St > St_c$ fractal clustering appears, while $St < St_c$ would lead to space filling properties according to his findings. In addition, his findings show a gradual change of the Lyaponov dimension. In this work, the simulations containing the ring prob-

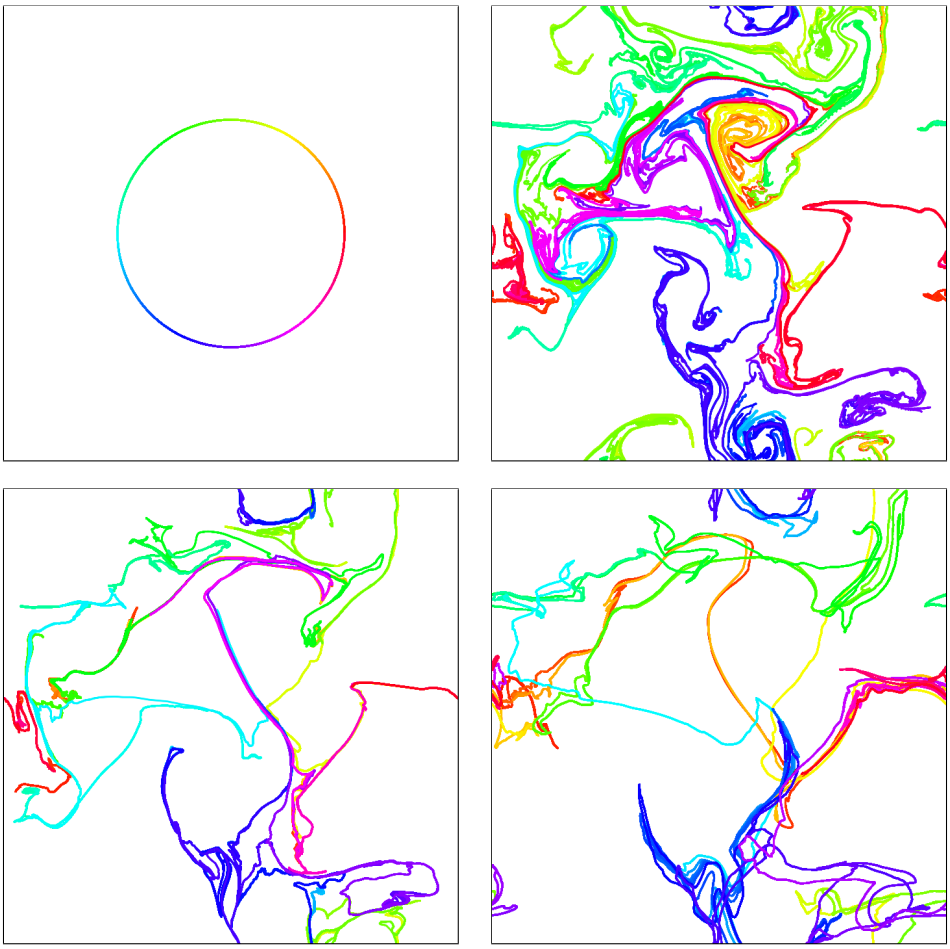

Figure 38: Particles initially arranged in a circle at time $t = 10$ (top, left) and advected at time $t = 20$ for $\tau_p = 0$ (top, right), $\tau_p = 0.5$ (bottom, left) and $\tau_p = 2$ (bottom, right)

lem are repeated for fluid particles embedded in turbulent flow. In addition, they are extended to inertial particles with different Stokes numbers. Later, the line

length and fractality are investigated. In order to avoid effects of the formation of isotropic turbulence from the initial conditions, the particles are embedded into the flow starting from $t = 10$. The previously studied initial condition is used for that and the time range was extended until $t = 30$ in order to have more deformation of the line. All particles where embedded in a circle with center (π, π) and diameter π. In total, four different particle sets were embedded. One as fluid particles with $\tau_p = 0$, and three with $\tau_p = 0.1$, $\tau_p = 0.5$ and $\tau_p = 2$. This should resemble particles with $St = 0$, $St < 1$, $St = 1$ and $St > 1$. For each set, 10 million particles were evenly distributed over the circle with no initial velocity.
The initial condition together with results at time $t = 20$ can be found in figure 38. The particles were coloured by initial angle in order to reference the course of the distorted ring at later times. Largely, three different behaviour emerges for different Stokes numbers. For flow with $St < 1$, the ring starts to be scattered along the computational domain. Vortices further and further distort the line especially close to vortex cores. In the special case of $St = 0$, the ring starts to fill up the entire domain. Overall, the line does not overlap itself for $St < 1$ and is only bend and distorted over time. For $St = 1$, the line does not start to fill up parts of the plane. Different sections of the line are moved close to each other and overlap in thin lines. Starting from $St > 1$, the line intersects with itself, which can clearly be seen in the lower right image of figure 38. Parts of the line oscillate around a common center, forming more structured-like behaviour.

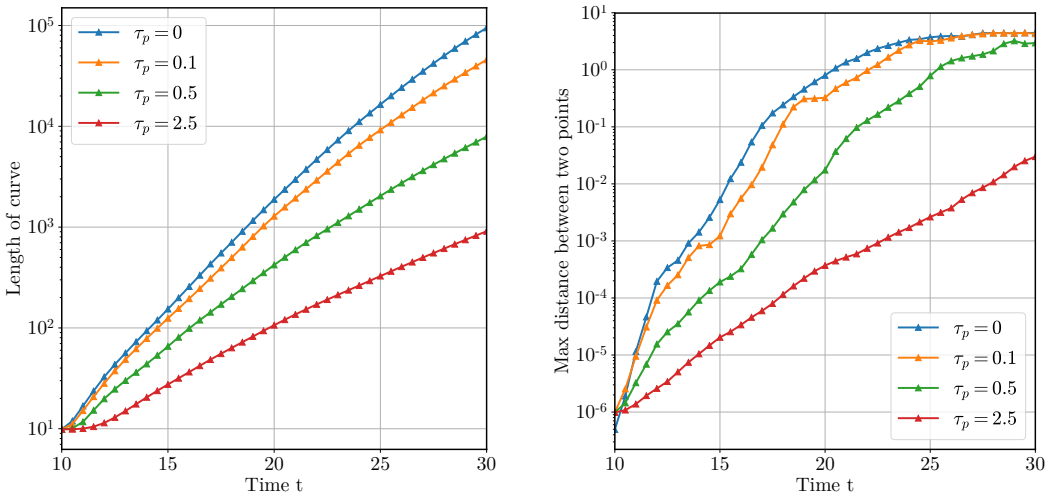

Figure 39: Length of the line (left) and maximum distance between two points (right) for the different particle sets.

The line length was computed by summing up the distance between all points. Results can be seen in figure 39. The greater the particle relaxation time, the greater the resistance is against distortion due to the turbulent flow, resulting in less line

elongation. Initially, heavier particles need more time in order to be accelerated. The line length experiences exponential growth for low times.
The left image of figure 39 shows the maximum distance between two points, which steadily increases over time for all different particle sets. Eventually it caps at π, which was used as a maximum length on the domain to regard periodicity. While this hints at large distortion levels for the individual particles, it was only achieved for a handful of points. This can be extracted from the cumulative density of the individual distances between two points, seen in the right image of figure 40. In fact, for all four particle sets, 99% of the distances between points was smaller than $1.2 \cdot 10^{-2}$. This can also be further shown by investigating the change in line length for different sampling rates of the points, effectively decreasing the number of particles, depicted in the left image of figure 40. While the length of the line does not seem to be completely converged, the change for small decreased sample rates is low. For increasing τ_p-values, this estimation becomes better and better. The particle density to represent the lines was therefore estimated to be sufficient for all particle sets.

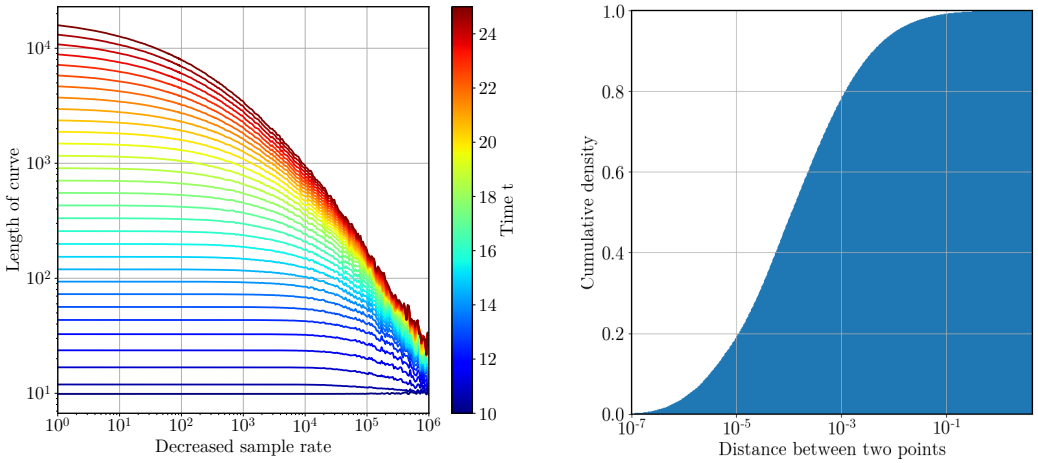

Figure 40: Length of the line with different sampling rates (left) and cumulative density for different distances between points at time $t = 30$ (right) for fluid particles.

7.5 Estimation of fractal dimension of ring of particles embedded in fully turbulent flow

Estimation of fractal quantities is generally a tedious task. Especially as the number of particles is finite, computing it comes with various challenges. In the scope of this work, first estimation of a Hausdorff dimension H_d will be presented, which were computed using a box counting algorithm. This algorithm computes the amount of uniform finite-sized boxes in the domain containing at least one particle over the total amount of boxes with varying box size. The Hausdorff dimension from this is defined as the rate of change for different box sizes.

In order to minimise local effects of the structure, a sliding box scan has been chosen. The size of the boxes is increased by a factor of 4 and the starting positions have been uniformly distributed so that the last box ends at the end of the domain. With this, at any position four boxes will overlap each other. This helps in detecting sharp edges for a given box size. It is especially helpful due to the fact, that the line is still only present at discrete points of the particles and small boxes will fail to appropriately capture the fractality of the actual lines.

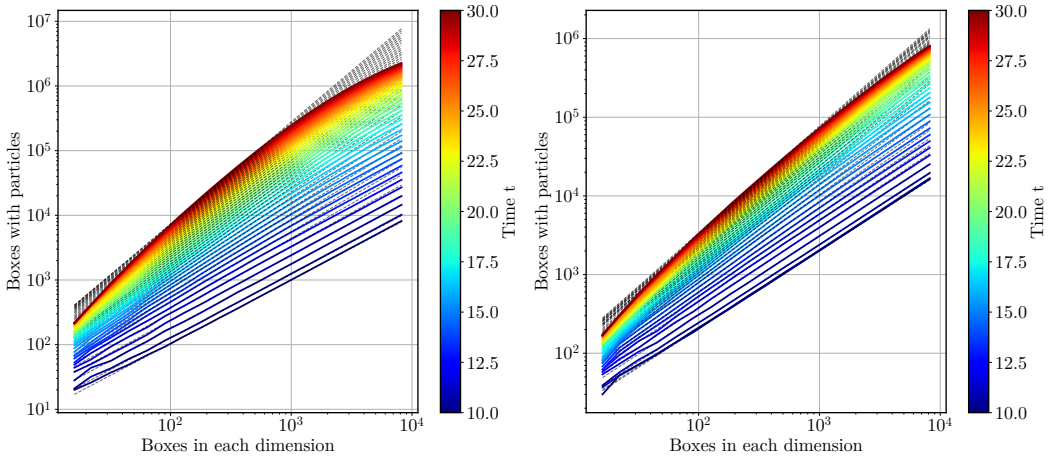

Figure 41: Boxes with particles over amount of boxes in each dimension over time for fluid particles (left) and particles with $St = 1$ (right). Estimated linear fits are included in black dotted lines.

The results for two particle sets over time can be found in figure 41. As expected, the amount of boxes with particles increases for all sets over time. In addition, the curves become further and further bent, as a small amount of boxes distributed over the region treats the scattered line segments as two-dimensional and a high amount of boxes starts to differentiate more and more between the individual particles. However, it can also be observed how this effect is different for each set of particles, with the particle set with $St = 1$ having more straight lines than the other sets.
In order to estimate a region suitable for linear fitting, the cumulative density was taken as a reference. It was chosen, that the box length should be larger than at least 90% of all distances between two points. This leads to a maximum of 775 boxes for the most affected particle set, which was chosen as an upper limit. For the lower limit, around 90 boxes was chosen. This would constitute of boxes having at maximum approximately $\frac{1}{20}$th of the domain length as box length. This was chosen to be sufficiently small to capture effects from the fractality of the curve.

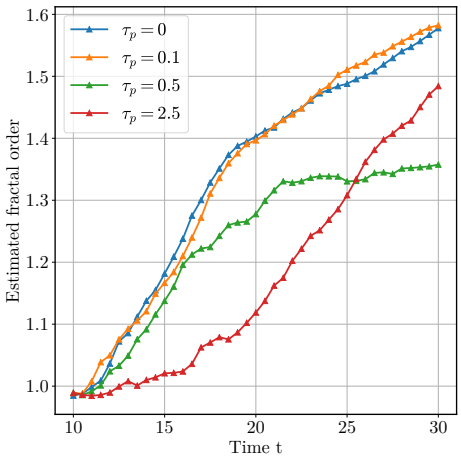

Figure 42: Estimated fractal order over time for ring problem with different particle sets.

The estimated fractal orders over time can be found in figure 42. It can generally be seen, that all fractal orders increase over time from a dimension of one, as the ring gets more and more distorted. Also, the fractal order for fluid particles and light particles of $St < 1$ increase at the same rate over time, which hints on their fractality being connected to each other. For particles with $St = 1$, the fractal order firstly increases, but stalls at around $H_d \approx 1.35$. This estimated dimension is the lowest in comparison to the other sets. However, against what was anticipated, a fractal dimension of one could not be confirmed. This could be due to some low vorticity regions present to form particle clusters, which increases the fractal order. For heavy particles with $St > 1$, their inertia initially resist the distortion of the line. At later times the estimated fractal dimension rises for later times though, approaching further and further the estimated value for the lighter particles.
Overall it can be seen, that while some first tendencies can be extracted from the results, they do not allow any clear conclusions. Mostly, because the time interval chosen did not allow for the ring of particles to completely spread out over the domain. For this, the initial condition has to be adapted in order to allow a longer range for isotropic homogeneous turbulence to be investigated. This can for example be achieved by increasing the number of vortices, leading to overall more mergers happening. However, this also increases the complexity and cost of the computations. In addition, further studies with varying random effects could be conducted. Mainly the change of position of the initial ring or random displacement of the Gaussian vortices.
Nevertheless, the fractal investigation could be a powerful tool in order to describe the characteristic behaviour of embedded particles in turbulent flow. A possible extension in 3D would be especially interesting, where the span of dimensions could be increased and particle sets could become completely space-filling, again forming line-like structures of dimension $H_D = 1$ or forming sheets of particles with dimension

$H_D = 2$.

7.6 Final state solution for isotropic turbulent flow with the characteristic mapping method

Two-dimensional decaying turbulent flow is known to approach apparently stable states after a long time evolution, according e.g. to [18]. For this section long term simulations with the initial conditions of the chequerboard pattern of 10×10 vortex cores were run. The simulation parameters were adapted in order to be more robust for a higher amount of remappings, they can be found in table 12. All variables for investigations where captured on a 4096^2 grid.

Name	Value	Name	Value
N_{coarse}	1024	N_{fine}	8192
N_ψ	4096	N_ω	8192
h_{fluid}	$1/3072$	$\delta_{inc,b}$	10^{-3}
ϵ_m	10^{-3}	k_{LP}	256
Fluid time scheme	RK3	Map update stencil	4th order

Table 12: Simulation parameters for the isotropic homogeneous turbulence simulation.

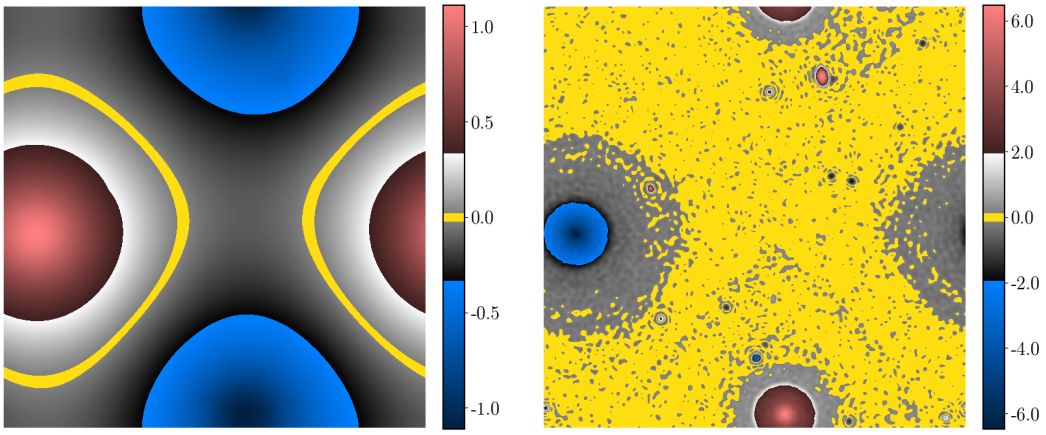

Figure 43: Stream function (left) and filtered vorticity (right) at time $T = 820$.

As the sub-maps after some time completely fill up the available CPU RAM of 190GB, the simulation was restarted several times with the latest vorticity possible sampled on the 4096^2 grid as a discrete initial condition. With that, the enstrophy conservation to the beginning was reduced due to interpolation effects to $2 \cdot 10^{-1}$. While this is very large, the continuations were able to preserve the patterns within the flow. The energy to the beginning was conserved up to a deviation of 10^{-2}. In

total, four simulations were run with a length of $[270, 255, 170, 125]$, summing up to a final time of $T = 820$.
As was shown by Segre et al. in [18], long term simulations for decaying 2D vorticity fields in periodic domains approach a stable state with two large vortices of opposing signs with the largest possible distance away from each other. This state is reached by continuous merging of the individual vortices from the initial condition. In fact, the merging of the vortices was already visible for the shorter version, however it will now be shown that for longer simulations eventually all vortices merge.

The results for the stream function and vorticity can be seen in figure 43. The

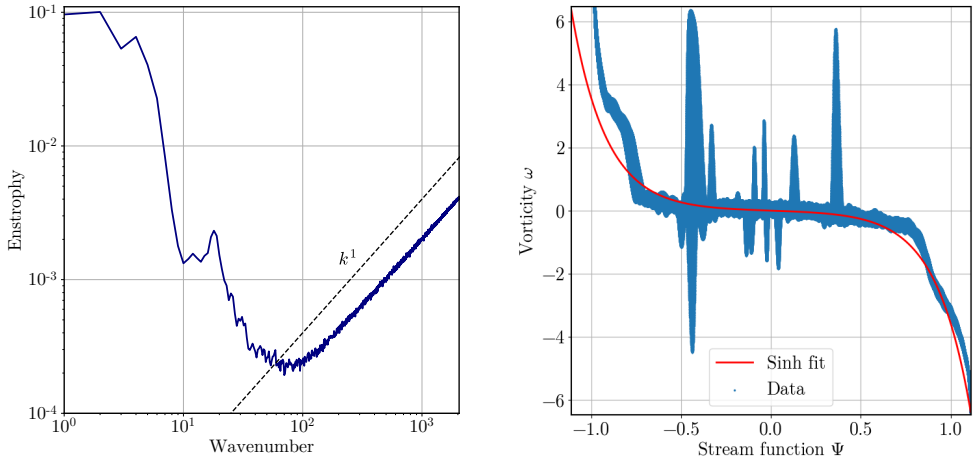

Figure 44: Enstrophy power spectrum (left) and correlation between vorticity and stream function (right) at $T = 820$.

vorticity shows large random oscillations as was therefore filtered with a low-pass filter of filter strength $k_{lp} = 64$ in order to show the global structures. This filter was chosen from the isotropic spectrum, depicted in the left graph of figure 44. There, the linear increase in enstrophy in larger scales is clearly visible, which experiences a k^1 power law. At around the filter strength the enstrophy is at minimum, so that all the artefacts are filtered out. The two main vortices are clearly visible, however, some minor other vortices are still present. The flow is therefore not fully merged yet and a prolonged simulation should give more stable results. However, a correlation between the stream function and the vorticity can already be established, seen in the right graph of figure 44. Besides some visible outliers coming from not yet merged vortices, most of the points follow a correlation. According to [17], this relation should approach a hyperbolic sine function. A fit with a least-square method, depicted in red, shows the functional relation. It was fitted to a function of $a \sinh(bx)$ with parameters $a \approx 0.04$ and $b \approx -5.21$. While the side with positive values of the stream function already largely follows the curve for large values, the negative part is not yet fully developed, mainly coming from the one missing larger vortex, which is also clearly visible within the graph. However it is important to

note, that especially the minor peaks and vortices take up only very little fraction of the domain in comparison to all the points close to the fitted curve. In [18] a similar initial condition with a checker-board of 8×8 vortices merged at a time of $T = 1291$, giving further evidence that the time of $T = 800$ was not yet able to depict the final time. Longer simulations which were out of the scope due to large CPU time requirements are thus expected to show a clear sinh functional relationship between the stream function Ψ and the vorticity ω. The characteristic mapping method is however able to reach the final states for the 2D incompressible Euler equations and show the functional relationship, as it can already be excerpted from the presented data.

8 Conclusion

The characteristic mapping method in its form presented in this thesis solves the 2D incompressible Euler equations by evolving the flow map along characteristic curves. With the gradien-augmented level set method and the computation of the Biot-Savart law in Fourier space, several mathematical building blocks have been presented that allow for an efficient discrete formulation and their efficient implementation on graphics processing units.

8.1 Summary

With the validations and investigations presented in this thesis, the framework currently serves as a well rounded up baseline in order to be used for further research applications. The mathematical features were implemented in a structured and clean way that allows for a lot of flexibility to adapt to specific needs in accuracy and efficiency, showcased in chapter 2 and 4. In space, several different grids are available in order to fine-tune the accuracy, RAM memory requirements for both CPU and GPU, and computational complexity. The grid of the flow sub-maps, called coarse grid, was found to be best chosen after the available CPU RAM available to fit all sub-maps. The fine grid, which is needed for the initial condition of the vorticity for each sub-map, can be chosen as large as possible for the available GPU RAM. In order to limit the amount of remappings for highly turbulent flow a low-pass filter can be applied to prolong the simulations. The grid of the stream function, called psi grid, was found to have a positive impact when increased, however the current implementation is not efficient. Utilising the Hermite interpolation, the convergence in space is of third order accuracy, consistent with the work of [21]. For the discretisation in time, several methods of different order of convergence have been implemented. This allows for first to fourth order convergence for the time-stepping methods, dependent on the need for accuracy or computational time. More refined time-stepping schemes RK3Mod and RK4Mod have been formulated for the given requirements, which also proved to be more efficient and should be chosen over their classical variants. Similar to [21], a Lagrange-interpolation of the velocity is used in order to use the Runge-Kutta schemes. Additionally the map update scheme for the update step of the gradient-augmented level set method has been investigated and increased to fourth order. Overall, all implementations for the discretisation in time were validated and yield the predicted order of convergence. Summing up, the framework does now feature a flexible and well-documented implementation of the characteristic mapping method for solving the 2D incompressible Euler equations. The used methods were adapted to reproduce the current convergence analyses in [21].
The implementation of point particles by Nicolas Saber in [15] was completed and validated in chapter 5. It was made consistent with the implementation of the time-stepping methods of variable convergence order. The methods were validated for both fluid and inertial particles in each a case with fixed velocity and fluid embed-

ded case with evolving flow map. It was found, that the more optimally chosen third and fourth order methods with less interpolations of velocity values deliver the best performance and are recommended for usage. Later on in section (7.3) the implementation of the particles was validated with respect to the literature for an estimated Stokes number. A particle relaxation time of $\tau_p \approx 0.5$ was found to correspond to $St = 1$. With this, further simulations can be done to investigate particle behaviour in turbulence, which is currently of high interest. The framework is able to quickly and precisely produce particle data, which can subsequently be used for analysis or comparison.
Also on the computer science part a lot has been done in order to make the whole code and framework more usable. GPU programming brings in many new programming concepts, of which the most important have been reported in chapter 3. This is especially useful, as mathematics and engineering scientists working with the code are not likely to be familiar with important programming concepts. In addition, the whole code has been reworked to be more understandable and usable. This features a large restructuring and commenting of the code including new function and source file structure. But also in order to use the code many aspects have been reworked as the extraction of variables to a settings file, the inclusion of parameters on the command-line level and usage of parameter files to enable detailed settings of all available parameters. The output structure was also reworked to be more consistent with options in the parameters to control in fine details which variables should be outputted at what times. Together with a short introduction for new users, the code should be ready to be further used by new scientists for their studies. In order to increase the availability, the code will be made open-source on GitHub withing the following weeks.
At the end of the thesis two applications were presented together with new physical features added to the code in the chapters 6 and 7. The inclusion of passive scalar fields with vanishing diffusivity in chapter 6 for turbulent mixing problems showed great potential for further research applications. As the computation of the advection of passive scalars does not impose any additional work, arbitrary many passive scalars can be included and investigated. This comes in handy especially with the zoom property of the characteristic mapping method, making it possible to investigate very fine details of emerging structures. In chapter 7 point particle behaviour in 2d turbulence was investigated. A first study on the dynamical behaviour gave further inside on chaotic behaviour and long term simulations are able to reach near final state solutions for relaxating vorticity fields. In both chapters two important initial conditions for theoretical studies, being the mixing layers and turbulence emerging from a random superposition of positive and negative vortices, have been further investigated and reworked for a more physical consistency.

8.2 Limitations and further research interests

The work in this thesis does not only provide a complete and finished project, but rather serves as a chapter within a larger research development.

The mathematical implementation now giving a working foundation can be further utilised. By far the most interesting addition would be the extension to three dimensions. This, however, would not only need the inclusion of new mathematical concepts from the Lie-algebra but also some restructuring of how the different dimensions are dealt with within the code. However, a documentation and first tests for the three-dimensional case have already been done in [22], so an implementation would be possible and straight-forward. This would open up the possibility for many new applications to be investigated, however memory issues could arise quickly. Besides that, the interpolation of the velocity could be changed for a Hermite interpolation, as done in [22] as well. This would stabilise the interpolation more but also increase the memory requirements for time-stepping methods of lower order.
For the code there is much that could still be done with many small improvements that could be made. They all represent a gradual change of the project from a C-like code to a more modern and object-oriented code with more features of C++. This would greatly reduce the amount of code, while increasing readability and flexibility. With the addition of vectors and classes for the variables, the whole process of transferring from and to the GPU and detailed index addressing could be greatly reduced. Already the usage of classes for the settings structure proved itself to be very useful for a flexible data structure, showcasing the potential. In addition, currently the largest bottleneck is in the Hermite interpolation of the velocity. While a fast implementation is present, it has to be improved in terms of memory handling. This could in the best case be a mere check on if the issues persist on new architectures and the worst case needs a manual implementation on how the data is loaded from the array to be interpolated from.
The currently available initial conditions and applications with the point particle implementation and possibility to compute passive scalars present a ready tool-set for further usage. There are several points that could be implemented. Especially with the forward flow map, forms of diffusion or external forces could be implemented for all passive elements. With the backwards and forward flow map data as a diffusion displacement could be tracked over time and be projected back onto the initial condition. Also the zoom property, being arguably the largest selling point of the characteristic mapping method in comparison to other methods, still lacks proper use-cases in order to show-case its potential.
Both the investigation of the dynamical behaviour of inertial particles 2D-turbulence and long time statistical turbulence, while being very promising aspects, did not deliver clear and interpretable results yet. Further refinement of the available initial condition can hopefully achieve more clean results.

8.3 Acknowledgements

The results of this work are part of an internship done at the Institut de Mathématiques de Marseille of the Aix Marseille Université in Marseille, France under Prof. Kai Schneider. It received funding from the ANR project CM2E (contract ANR-20-

CE46-0010) together with an Erasmus+ grant, which is thankfully acknowledged. I would also like to thank Prof. Kai Schneider and Thibault Oujia from the I2M research group for the opportunity to do this internship in Marseille and for their support during that time. I also want to thank Prof. Julius Reiss for his support from the Technische Unversität Berlin.

9 Appendix

9.1 Convergence order in time for all time-stepping schemes

The convergence behaviour in time for all first to fourth order time-stepping schemes will be given here. A further explanation of all used parameters can be found in section (4.3.2). The simulations where done with a Lagrange interpolation of the velocity with a convergence order respectant to the order of the time-stepping scheme used.

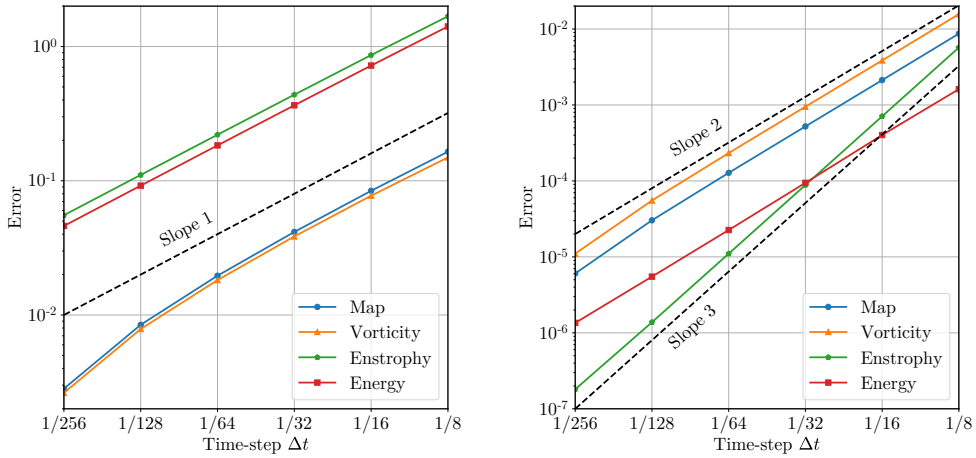

Figure 45: Convergence errors in time with first order Euler explicit (left) and second order Heun (right) methods.

For the first order Euler explicit method, the error quantities are all of first order. Surprisingly, the enstrophy and energy error are way higher than the other two error quantities. Due to the high error, a first order implementation is not practical for usual simulations and is mainly used for testing purposes.
The second order Heun's method shows largely second order convergence behaviour. However, surprisingly, the Enstrophy shows third order accuracy.

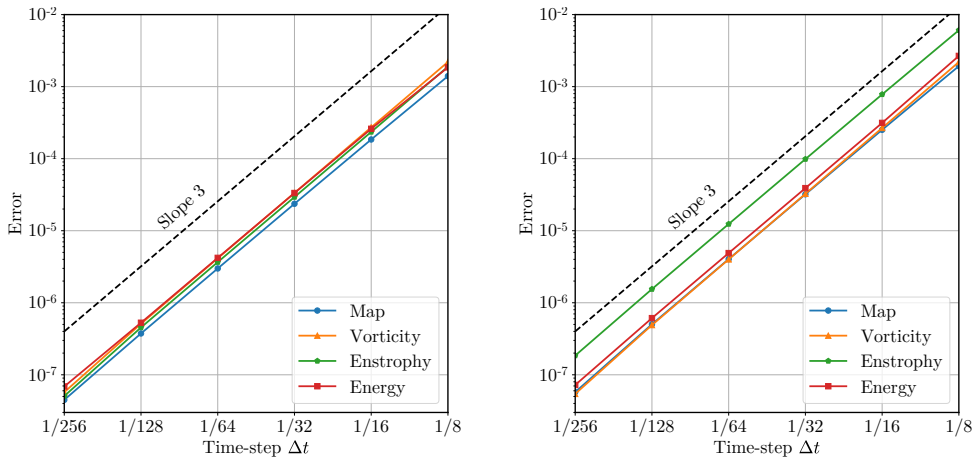

Figure 46: Convergence errors in time with classical (left) and modified (right) third order Runge-Kutta methods.

The two third order methods show very similar behaviour. Both show third order convergence for all quantities. The modified version shows slightly less convergence, especially for the enstrophy.

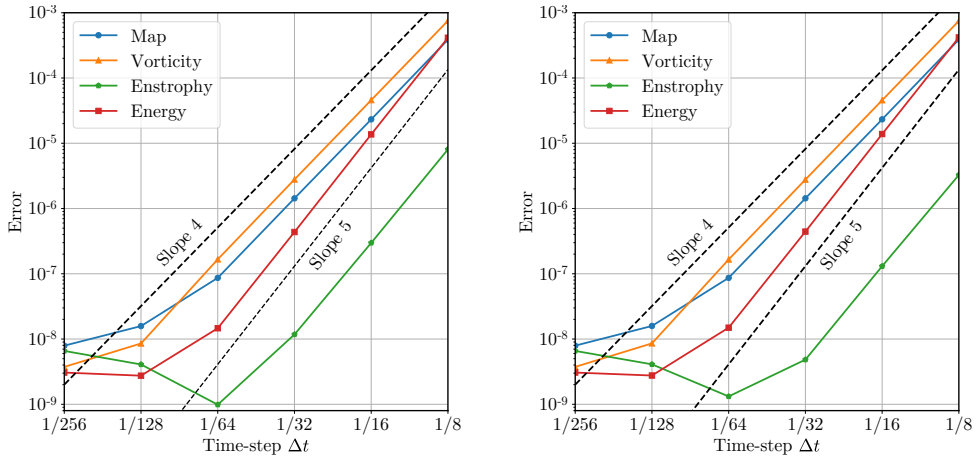

Figure 47: Convergence errors in time with classical (left) and modified (right) fourth order Runge-Kutta methods.

Both fourth order methods further decrease the error. However, only the map and vorticity error are of fourth order accuracy, while the conservation of energy and enstrophy is further improved to show fifth order accuracy. It is not clear why this is the case. At an error of around 10^{-8} all error quantities stagnate and reach a minimum.

9.2 Convergence order in time for different order of Lagrange polynomials

In comparison to the convergence graphs with second and third order of Lagrange polynomials in section (4.3.2), the convergence properties with first and fourth order implementation are given in the subsequent graphs.
The values for first order are as expected, with first order behaviour for all quantities except the conservation of enstrophy. For fourth order, the order of convergence stays is not further increased with energy and enstrophy error being very similar to the third order implementation. Surprisingly however, the higher order Lagrange interpolation is able to decrease the map and vorticity error by a noticeable amount, which is roughly a factor of 8.

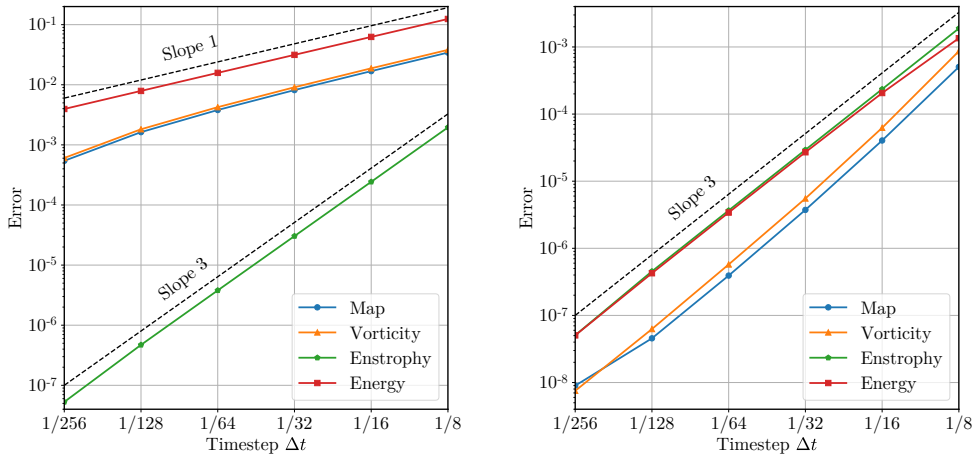

Figure 48: Convergence errors in time with first order (left) and fourth order (right) Lagrange interpolation of the velocity.

References

[1] J. R. Bates and A. McDonald. "Multiply-Upstream, Semi-Lagrangian Advective Schemes: Analysis and Application to a Multi-Level Primitive Equation Model". *Mon. Weather Rev.* 110.12 (Jan. 1982), p. 1831. DOI: 10.1175/1520-0493(1982)110<1831:MUSLAS>2.0.CO;2.

[2] Jérémie Bec. "Fractal clustering of inertial particles in random flows". *Phys. Fluids* 15.11 (2003), pp. L81–L84. DOI: 10.1063/1.1612500.

[3] Henning Bockhorn, Wolfgang Gerlinger, Kai Schneider, and Jörg Ziuber. "Simulation and Analysis of Mixing in Two-Dimensional Turbulent Flows Using Fourier and Wavelet Techniques". In: *Scientific Computing in Chemical Engineering II*. Ed. by Frerich Keil, Wolfgang Mackens, Heinrich Voß, and Joachim Werther. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 344–351. ISBN: 978-3-642-60185-9. DOI: 10.1007/978-3-642-60185-9_40.

[4] J. C. Butcher. "Runge–Kutta Methods". In: *Numerical Methods for Ordinary Differential Equations*. J. C. Butcher, 2016. Chap. 3, pp. 143–331. ISBN: 9781119121534. DOI: 10.1002/9781119121534.ch3.

[5] René A. Carmona and Frederic Cerou. "Transport by incompressible random velocity fields: simulations & mathematical conjectures". In: *Stochastic Partial Differential Equations: Six Perspectives*. Ed. by René A. Carmona and Boris Rozovskii. Vol. 64. Mathematical Surveys and Monographs, 1999, pp. 153–181. ISBN: 978-1-4704-2853-2.

[6] René A. Carmona, Stanislav A. Grishin, and Stanislav A. Molchanov. "Massively Parallel Simulations of Motions in a Gaussian Velocity Field". In: *Stochastic Modelling in Physical Oceanography*. Ed. by Robert J. Adler, Peter Müller, and Boris L. Rozovskii. Boston, MA: Birkhäuser Boston, 1996, pp. 47–68. ISBN: 978-1-4612-2430-3. DOI: 10.1007/978-1-4612-2430-3_2.

[7] Prince Chidyagwai, Jean-Christophe Nave, Rodolfo Ruben Rosales, and Benjamin Seibold. *A comparative study of the efficiency of jet schemes*. 2012. arXiv: 1104.0542 [math.NA].

[8] H.J.H Clercx, A.H Nielsen, D.J Torres, and E.A Coutsias. "Two-dimensional turbulence in square and circular domains with no-slip walls". *Europ. J. Mech. - B/Fluids* 20.4 (2001), pp. 557–576. ISSN: 0997-7546. DOI: 10.1016/S0997-7546(01)01130-X.

[9] *CUDA C++ Programming Guide*. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html. Accessed: 2022-04-20.

[10] Marcel Lesieur. *Turbulence in fluids: Stochastic and numerical modelling*. Vol. 6. Mechanics of Fluids and Transport Processes. Springer, 1987. ISBN: 978-9-4009-3545-7.

[11] Keigo Matsuda, Kai Schneider, and Katsunori Yoshimatsu. "Scale-dependent statistics of inertial particle distribution in high Reynolds number turbulence". *Phys. Rev. Fluids* 6 (6 June 2021), p. 064304. DOI: 10.1103/PhysRevFluids.6.064304.

[12] Martin R. Maxey and James J. Riley. "Equation of motion for a small rigid sphere in a nonuniform flow". *Phys. Fluids* 26.4 (1983), pp. 883–889. DOI: 10.1063/1.864230.

[13] Olivier Mercier, Xi-Yuan Yin, and Jean-Christophe Nave. *The Characteristic Mapping Method for the Linear Advection of Arbitrary Sets*. 2013. arXiv: 1309.2731 [math.NA].

[14] Jean-Christophe Nave, Rodolfo Ruben Rosales, and Benjamin Seibold. "A gradient-augmented level set method with an optimally local, coherent advection scheme". *J. of Comput. Phys.* 229.10 (2010), pp. 3802–3827. DOI: 10.1016/j.jcp.2010.01.029.

[15] Nicolas Saber. "Two-dimensional Characteristic Mapping Method with inertial particles on GPU using CUDA". MA thesis. France: Aix-Marseille University, 2021.

[16] Kai Schneider and Marie Farge. "Numerical simulation of a mixing layer in an adaptive wavelet basis". *Comptes Rendus de l'Académie des Sciences - Series IIB - Mechanics-Physics-Astronomy* 328.3 (2000), pp. 263–269. ISSN: 1287-4620. DOI: 10.1016/S1287-4620(00)00106-X.

[17] Kai Schneider, Marie Farge, and Nicolas Kevlahan. "Spatial Intermittency in two-dimensional Turbulence: A Wavelet Approach". In: *Woods Hole Mathematics*, pp. 302–328. DOI: 10.1142/9789812701398_0007.

[18] Enrico Segre and Shigeo Kida. "Late states of incompressible 2D decaying vorticity fields". *Fluid Dyn. Res.* 23.2 (Aug. 1998), pp. 89–112. DOI: 10.1016/S0169-5983(97)00050-6. arXiv: chao-dyn/9709020 [nlin.CD].

[19] Badal Yadav. "Characteristic Mapping Method for Incompressible Euler Equations". MA thesis. Canada: McGill University, 2015.

[20] Xi-Yuan Yin, Linan Chen, and Jean-Christophe Nave. "A Diffusion-Driven Characteristic Mapping Method for Particle Management". *SIAM J. Sci. Comput.* 43.5 (2021), A3155–A3183. DOI: 10.1137/20M1364357.

[21] Xi-Yuan Yin, Olivier Mercier, Badal Yadav, Kai Schneider, and Jean-Christophe Nave. "A Characteristic Mapping method for the two-dimensional incompressible Euler equations". *J. of Comput. Phys.* 424 (2021), p. 109781. DOI: 10.1016/j.jcp.2020.109781.

[22] Xi-Yuan Yin, Kai Schneider, and Jean-Christophe Nave. *A Characteristic Mapping Method for the three-dimensional incompressible Euler equations*. 2021. arXiv: 2107.03504 [math.NA].