Numerical methods for set transport and related partial differential equations

Olivier Mercier

Master of Science

Department of mathematics and statistics

McGill University

Montreal, Quebec

June 2013

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Master of Science in Mathematics and Statistics

ACKNOWLEDGEMENTS

I would like to thank my supervisor Prof. Jean-Christophe Nave for his academic and moral support throughout my master program. I would also like to thank the Natural Sciences and Engineering Research Council of Canada and the Fonds québécois de la recherche sur la nature et les technologies for their financial support.

The new ideas presented in chapter 3 were developed in collaboration with Jean-Christophe Nave, Rodolfo Ruben Rosales (Massachusetts Institute of Technology) and Benjamin Seibold (Temple university). The new method of chapter 4 was developed with Jean-Christophe Nave. Applications to Euler equations in chapter 5 were done with Jean-Christophe Nave and Kai Schneider (Université de Provence).

ABSTRACT

In many cases, the simulation of a physical system requires to track the evolution of a set. This set can be a piece of cloth in the wind, the boundary between a body of water and air, or even a fire front burning through a forest. From a numerical point of view, transporting such sets can be difficult, and algorithms to achieve this task more efficiently and with more accuracy are always in demand. In this thesis, we present various methods to track sets in a given vector field. We also apply those techniques to various physical systems where the vector field is coupled to the advected set in a non-linear way.

ABRÉGÉ

Dans plusieurs situations, la simulation de systèmes physiques requiert de suivre l'évolution d'un ensemble. Cet ensemble peut être un bout de tissu dans le vent, la frontière entre une masse d'eau et l'air, ou même le front d'un feu brûlant à travers une forêt. D'un point de vue numérique, transporter de tels ensembles peut être difficile, et des algorithmes pour accomplir cette tâche plus efficacement et avec plus de précision sont toujours en demande. Dans ce mémoire, nous présentons plusieurs méthodes pour suivre l'évolution d'ensembles dans un champ de vecteur donné. Nous appliquons aussi ces techniques à divers systèmes physiques où le champ vectoriel est couplé de manière non linéaire aux ensembles évolués.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
Introduction

A wide variety of scientific problems have as their main objective the evolution of a set. In hydrodynamics, the set can be the interface between water and air. In computer graphics, the set can be part of a smooth surface representing a piece of cloth. In meteorology, the set can be the collection of water droplets forming a cloud. In all of these applications, we are interested in knowing how the set evolves under given physical laws.

These settings give rise to systems of partial differential equations (PDE). For most interesting applications, the equations governing the motion of the abovementioned sets is only a piece of a very complex puzzle. Other equations are included in the system to describe the evolution of all the variables in the physical situation of interest. But still, the final goal is often to compute the evolution of the set, as is the case in the three examples given above.

The system of differential equations involved are usually too complicated to be solved analytically, and numerical methods have to be employed. This thesis thus focuses on numerical methods to transport sets. More precisely, we focus on the problem of a set passively transported in a vector field, as depicted in figure 1–1. Such a vector field is often physically determined by the other equations of the system, but in some cases the vector field is provided explicitely. This is for example useful in artistic simulations where an artist wants to dictate the movement of the set.

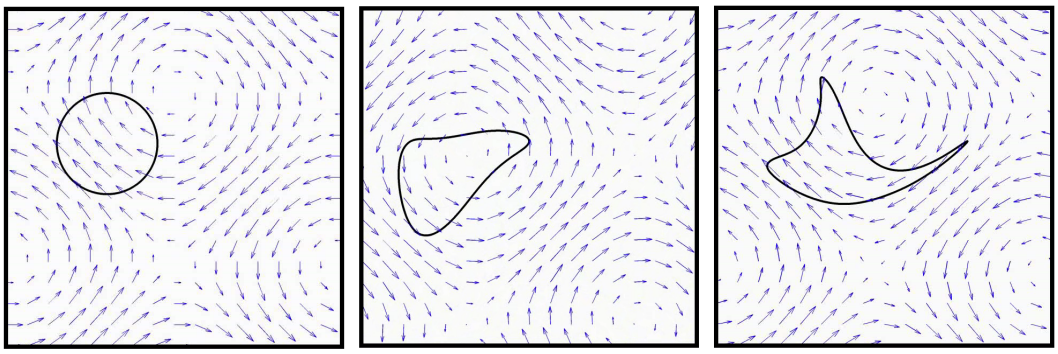


Figure 1–1: Example of set transport. An initial circle is deformed under a time dependant velocity field.

This thesis is organized as follows. Chapter 2 presents the level set method, a widely used approach to transport closed manifolds such as closed curves in 2D or surfaces in 3D. Chapter 3 explores different techniques using particle clouds and their combination with level set methods. Then in chapter 4 we consider the advection of general sets through the use of diffeomorphisms. All those methods are put to good use in chapter 5, where we apply them to the complicated and non-linear case of fluid simulations. Finally, chapter 6 summarizes the thesis and presents some outlooks and future work.

CHAPTER 2
Level set methods

In many applications, the set to be evolved is a closed manifold, like the boundary of a body of water, or a flame front burning through a forest. The location of the set is of course the most important aspect of the manifold that we want to track, but other features are of interest. For instance, it is useful to know which part of the domain is enclosed by the manifold, and which part is outside. This can determine which part of the forest has alreadry burnt, and which part is yet untouched. We can also be interested in the normal vector at points of the manifold. This is useful to know how pressure acts in a fluid simulation, for instance.

Level set methods allow to advect manifolds while representing all those features, and even more. The method was first introduced by Osher and Sethian in 1988 [20]. The idea they introduced comes from the implicit representation of closed manifolds, that is, the representation through an implicit function in higher dimension. For example, to describe a unit circle centered at the origin of the plane, a simple equation is

$$x^2 + y^2 - 1 = 0. \tag{2.1}$$

We say that the circle is the *zero level set* of the function $\phi(x, y) := x^2 + y^2 - 1$. In general, in d dimensions, the level set α of a function $\phi(\vec{x}) : \mathbb{R}^d \to \mathbb{R}$ is the set satisfying $\phi(\vec{x}) = \alpha$. Throughout this thesis, the level set $\alpha = 0$ will be used unless stated otherwise.

Similarly, when given a closed manifold M, any function ϕ such that M is a level set of ϕ is called a *level set function* for M. Note that for a given manifold M, the level set functions is not unique. For instance, if the manifold is again the unit circle centered at the origin, we can use $\phi(x, y) = x^2 + y^2 - 1$ as above, but $\psi(x, y) = \sqrt{x^2 + y^2} - 1$ also works. We have that

$$\psi(x, y) = 0 \tag{2.2}$$

$$\Leftrightarrow \quad \sqrt{x^2 + y^2} = 1 \tag{2.3}$$

$$\Leftrightarrow \quad x^2 + y^2 - 1 = 0 \tag{2.4}$$

so the unit circle is indeed a zero level set of the function ψ. For ease of visualization, the two functions and their identical zero level sets are shown in figure 2–1. A choice of level set function therefore has to be made, and the possibilities are infinite. Still, some level set functions are better than others from a numerical point of view. To see this, we look at three different level set functions representing the same 1D interval $[-1, 1]$. Those functions, shown in black from left to right in figure 2–2, are

$$\phi_1(x) = 0.075x^2 - 0.075 \tag{2.5}$$

$$\phi_2(x) = |x| - 1 \tag{2.6}$$

$$\phi_3(x) = \arctan\left(x^{100}\right) - \frac{\pi}{4}. \tag{2.7}$$

In red is the graph of the same functions shifted up by 0.05 to imitate numerical errors. Also, under each graph is a figure showing the zero level sets of the functions. The black rectangle represents the $[-1, 1]$ interval, and the red bars represent the bounds of the interval represented by the shifted level set functions.

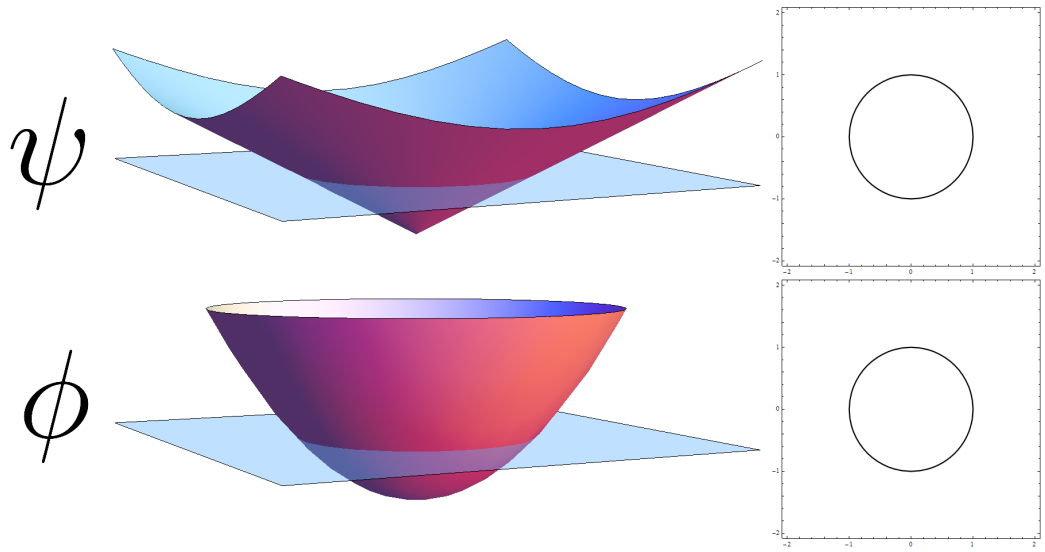


Figure 2–1: Two level set functions for the same circle. The plane $z = 0$ intersects both functions and the slice is shown to the right of each function. The zero level set is indeed the same.

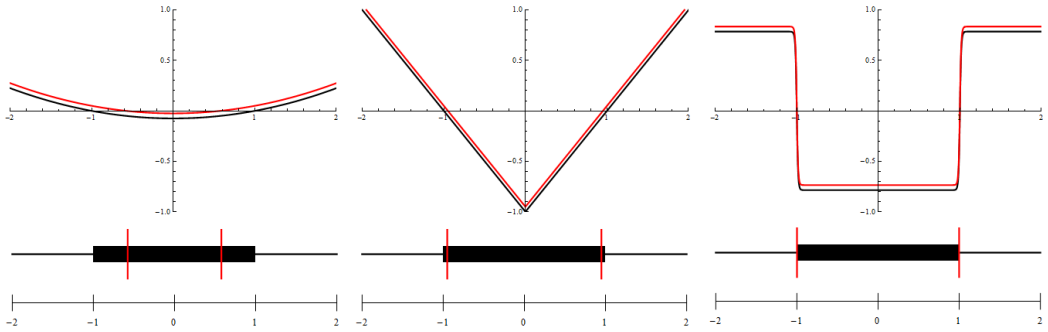


Figure 2–2: Three perturbed level set functions. All initial functions (black) represent the same interval, and they are shifted up by 0.05 to imitate numerical errors. Under each function is the set they represent (red) compared to the [-1,1] interval (black rectangle). A steeper slope is more resistant to perturbations.

From this example, we see that the steeper the level set function is around its zero level set, the more robust the corresponding set is to perturbations. In a real life situation, various numerical errors will add noise to the level set function, so it is essential to ensure robustness. Unfortunately, a very steep function such as ϕ_3 is difficult to represent numerically. We will discuss this in more details in section 2.2, but the basic argument is that level set functions are often represented by polynomials. Since polynomials are continuous and smooth, they are not well suited to represent almost discontinuous functions such as ϕ_3. Therefore, when choosing the level set function to represent a set, a good compromise is to aim for a function with unit slope. Formally, we choose the level set function ϕ so that it satisfies

$$\begin{cases} M = \{\vec{x} \in \mathbb{R}^2 \mid \phi(\vec{x}) = 0\} \\ ||\nabla\phi|| = 1 \ \text{ almost everywhere} \\ \text{interior}(M) = \{\vec{x} \in \mathbb{R}^2 \mid \phi(\vec{x}) < 0\} \end{cases} \tag{2.8}$$

where ∇ is the gradient operator. Such a definition gives a unique level set function for any given smooth manifold M.

An example of a level set function that satisfies definition (2.8) for the unit circle is the function $\psi(x, y) = \sqrt{x^2 + y^2} - 1$ seen previously in figure 2–1. The gradient is 1 everywhere except at the center of the circle, which is in accordance with definition (2.8). In general, such a level set function can be defined as the signed distance function of the set.

Now, remember that our objective is to transport sets in a vector field. The main advantage of the level set formulation is that set advection translates to a simple PDE for the level set function. Given a vector field $\vec{u}(\vec{x}, t)$, we want every point of the set to move with the direction and velocity prescribed by

the vector field at any time. Let $M(t)$ be the transformed set at time t, with $M(0)$ being the initial set. We want

$$\partial_t \vec{x}(t) = \vec{u}(\vec{x}(t), t) \tag{2.9}$$

for all points $\vec{x}(t) \in M(t)$ for all times t. Equation (2.9) thus describes the path that points $\vec{x}(t)$ follow in the vector field. Such trajectories are called *characteristic curves*.

In the level set formulation, we want to modify the level set function $\phi(\vec{x}, t)$ so that its zero level set is $M(t)$ at all time, that is

$$\phi(\vec{x}(t), t) = 0 \quad \forall \vec{x}(t) \in M(t) \quad \forall t. \tag{2.10}$$

But since the level set function is defined everywhere, and not only on the zero level set, we extend this condition so that it applies to all level sets. Let ϕ_0 be the level set function at time $t = 0$, we want

$$\phi(\vec{x}(t), t) = \phi_0(\vec{x}(0)) \quad \forall \vec{x} \quad \forall t. \tag{2.11}$$

That is, we want the values of ϕ to be constant on characteristic curves. Taking a time derivative of (2.11) and using the chain rule, we get

$$\partial_t \phi(\vec{x}(t), t) + \partial_t \vec{x}(t) \cdot \nabla \phi(\vec{x}(t), t) = 0 \tag{2.12}$$

which, combined with equation (2.9), becomes

$$\partial_t \phi(\vec{x}(t), t) + \vec{u}(\vec{x}(t), t) \cdot \nabla \phi(\vec{x}(t), t) = 0. \tag{2.13}$$

We pair this equation to the initial condition $\phi(\vec{x}(0), 0) = \phi_0(\vec{x})$ to have a closed system of differential equations. For clarity, we will drop the arguments of most functions from now on. Doing so, equation (2.13) and its initial condition

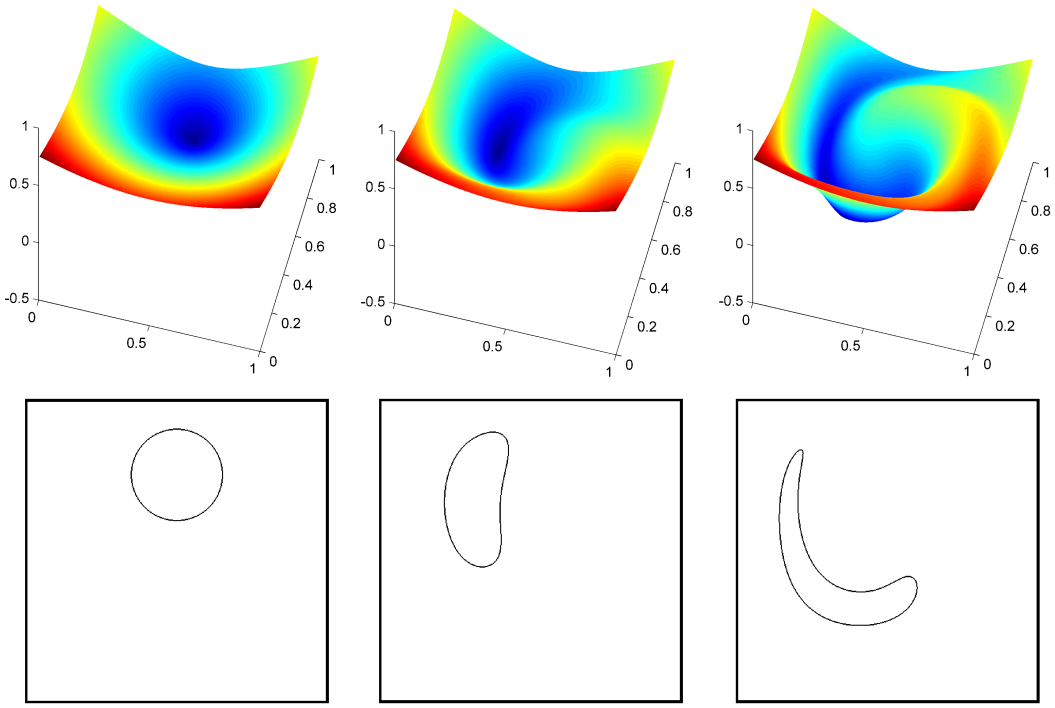


Figure 2–3: Example of evolution of a level set function as a solution of equation (2.14). The zero level set is represented under the level set functions. The scale blue to red goes from negative to positive values.

become

$$\begin{aligned} \partial_t \phi + \vec{u} \cdot \nabla \phi &= 0 \\ \phi(\vec{x}, 0) &= \phi_0(\vec{x}). \end{aligned} \tag{2.14}$$

This last set of equations is called the *linear transport equation*. An example of its solution is shown in figure 2–3, where we see the initial level set function being evolved under some vector field. The corresponging zero level sets are also shown under each functions.

Level set methods are therefore very convenient since they completely abstract from the set itself by transfering all the information to the level set function. One important consequence is that the topology of the transported sets can change over time, and this change is completely oblivious to the method. For instance, figure 2–4 uses two disjoint circles for an initial set. The

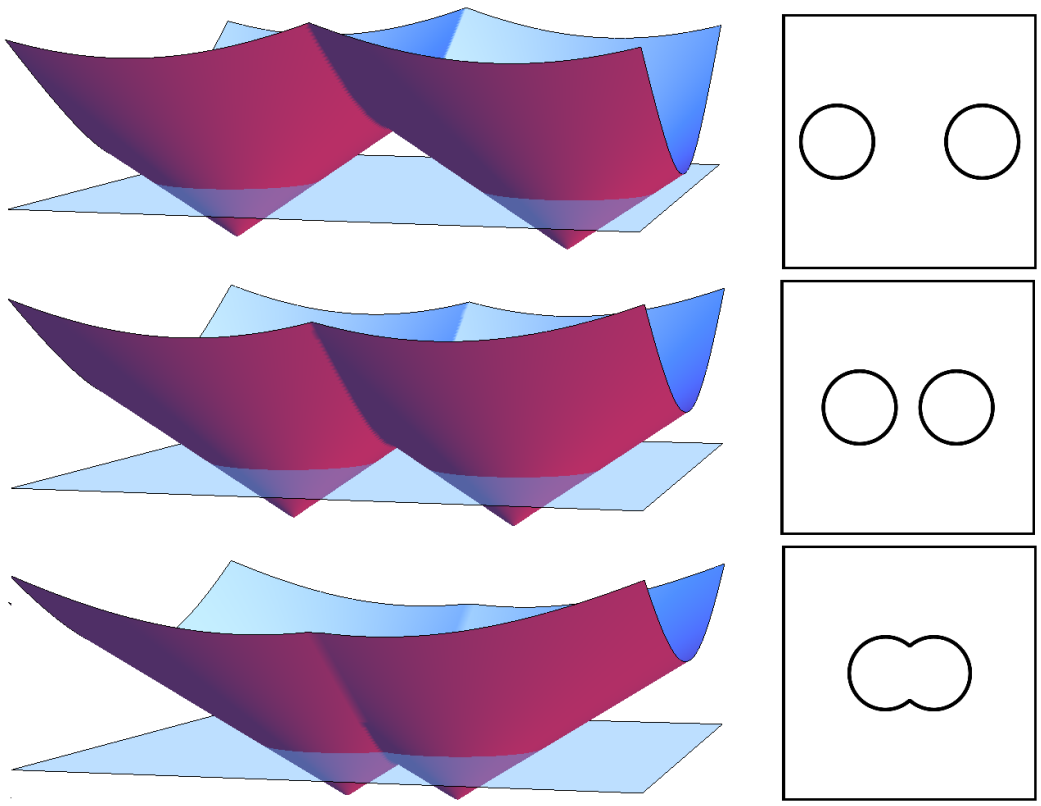


Figure 2–4: Level set function and topology changes. The zero level set initially has two components, which merge into one. This change is transparent to the level set function.

velocity field then makes the two circles collapse, which changes the topology. But this is not noticeable when only looking at the level set functions. Being able to easily deal with such changes is one of the main reasons the level set methods are so popular, because topology changes are very common in practical applications. Consider for instance the simulation of a drop of water entering a pool, where two disjoint bodies become one.

The transport equation (2.14) is well known and well understood. It is a hyperbolic PDE whose solution is given entirely from its characteristic curves and its initial condition. As stated by equation (2.11), the solution along characteristic curves is constant. Therefore, to compute the value of $\phi(\vec{x}, t)$, we only have to track the point $\vec{x}$ backwards alongs its characteristic curve up

to time $t = 0$. At this time, the solution is known (ϕ_0) and the value at time t can be evaluated. The next section focuses on computing those characteristic curves.

2.1 Computing characteristic curves

We stated in the previous section that the solution of the transport equation (2.14) reduces to the computation of characteristic curves. To compute the characteristic curve corresponding to a given point $\vec{x}_0$ at time t_0, we need to solve

$$\begin{aligned} \partial_t \vec{x(t)} &= \vec{u}(\vec{x}(t), t) \\ \vec{x}(t_0) &= \vec{x}_0. \end{aligned} \tag{2.15}$$

This is a simple ordinary differential equation (ODE). Depending on $\vec{u}$, it may be solvable analytically, but in general numerical methods have to be used. When a point is tracked back in time following a characteristic curve, the result is called a *footpoints* of this point, and will be denoted by $\mathring{x}(\vec{x}, t)$. Note that there is a time dependence since the location of the footpoint depends on how far back we track the point $\vec{x}$ in time.

A popular family of methods to solve ODEs are the Runge-Kutta methods. They can be found in most numerical analysis book, such as [3]. To give a quick overview of those methods, let's look at the first order Runge-Kutta methods, also known as the Euler method. Starting at our initial condition (i.e. time $t = t_0$ and position $\vec{x}_0$), we want to know what the solution is a small time away from t_0. Since we are interested in solving the characteristics backwards in time, we want to compute the footpoint at time $t_0 - \Delta t$ for some small time interval Δt. The Euler method approximates the solution by

$$\vec{x}(t_0 - \Delta t) \approx \vec{x}(t_0) - \Delta t \; \vec{u}(\vec{x}_0, t_0). \tag{2.16}$$

By taking a Taylor expansion of the left hand side of (2.16), we can make the more precise statement

$$\vec{x}(t_0 - \Delta t) = \vec{x}(t_0) - \Delta t\ \vec{u}(\vec{x}_0, t_0) + O(\Delta t^2). \tag{2.17}$$

This means that the error we make using this method is, for asymptotically small values of Δt, proportionnal to Δt^2. For this reason, we say the method is *locally second order*[1] .

Runge-Kutta methods are generalized versions of the Euler method which provide schemes of higher order. These methods can be used to solve ODE (2.15), but only for a small time step Δt. To solve the ODE for a longer time, we iterate; Because of the initial condition in equations (2.15), the solution is known at time $t = t_0$, so we can use a Runge-Kutta method to find the solution at $t_0 - \Delta t$. With this new information, we can compute the solution at $t = (t_0 - \Delta t) - \Delta t = t_0 - 2\Delta t$, and so on.

Depending on the vector field $\vec{u}$, the characteristic curves can have singular behaviors [7]. Figure 2–5 shows three different cases regarding characteristics. On the left, we have the simple case where two characteristics ending at different points do not cross in the past. This is the standard behavior for the linear transport equation with regular enough $\vec{u}$. Then, the middle picture shows the case where characteristics form a *shock*, that is, two characteristics

[1] Numerical schemes for ODEs are usually classified by their global order instead of their local order. The global order is the accumulated error made to reach a certain time, while the local order is the error made at every time step. The global order is always one less than the local order for ODEs. This explains, for instance, the nomenclature of Runge-Kutta methods. See [3] for more details.

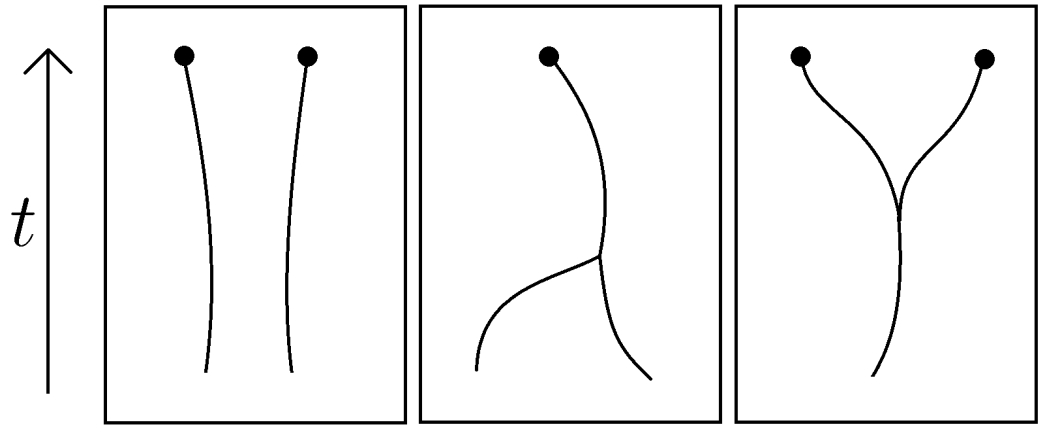


Figure 2–5: Three different behaviors of characteristic curves. The left figure is the regular case where the characteristic curves do not cross, the middle picture shows a shock, and the last figure shows a rarefaction.

join at some point. This is a problem, because when tracing a characteristic backwards, the curve splits into multiple branches and a choice has to be made. The third case is a *rarefaction*, where a characteristic curve splits into two. When tracing curves backwards, this causes two points to transport the same information, and can cause irregular behavior in the solution. For now, we assume that only the first case is present, i.e., the characteristic curves do not cross. This problem will resurface when we look at fluid simulations in chapter 5.

Using an ODE solver, we can devise an algorithm to solve the PDE (2.14) approximately. It is described in algorithm 1. Of course, this algorithm is not

Algorithm 1 Solving the transport equation

· Define the initial level set function ϕ_0 representing the initial set M.
for $t = \Delta t$ to ∞ **do**
  · For all points in the domain, compute the footpoint $\mathring{x}$ at time $t - \Delta t$.
  · Evaluate $\phi(\mathring{x}, t - \Delta t)$. Copy this value to $\phi(\vec{x}, t)$.
  · Increase t by Δt.
end for

usable from a numerical point of view, because it requires at every time step to evaluate the new level set function at every single point in the domain. The

next section explains how to solve this problem by only evaluating the function at a finite set of points.

2.2 Interpolation spaces

Since we cannot evaluate the evolved level set function at every point in algorithm 1, we need to use interpolation. There are countless interpolation spaces that have been developed, and the best choice usually depends on the situation. For instance, if we know a priori that the function to approximate is monotone, it is wise to pick an interpolant that guarantees this property. In this section, we focus on two interpolants that can be used in a wide variety of situations.

The interpolants we present are based on regular grids. For clarity, we suppose the whole domain of the problem is the 2D square $[0,1] \times [0,1]$. Nevertheless, the methods are easily generalizable to any dimension and can be adapted to deal with any domain shape.

For a given integer n, we call *grid points* the set of $(n+1) \times (n+1)$ points

$$\left\{ (x_i, y_j) := \left(\frac{i}{n}, \frac{j}{n} \right) \middle| \, i, j \in \{0, 1, ..., n\} \right\} \tag{2.18}$$

and we index them using the values i and j of their definition. The value of a function ϕ on a grid point will be denoted using the same index, i.e. $\phi_{i,j} := \phi(x_{i,j})$, and similarly for any derivatives of the function. We call *grid* the graph that joins every two grid points for which all indices are the same except for one that differs by exactly one. We say that a grid with grid points (2.18) has *size* $n \times n$. For example, an 8×8 grid is shown in figure 2–6. Note that such a grid divides the domain into multiple regions called *cells*. In 2D, these cells are indexed by (i, j) where $i, j \in \{1, ..., n\}$. The cell (i, j) is the one

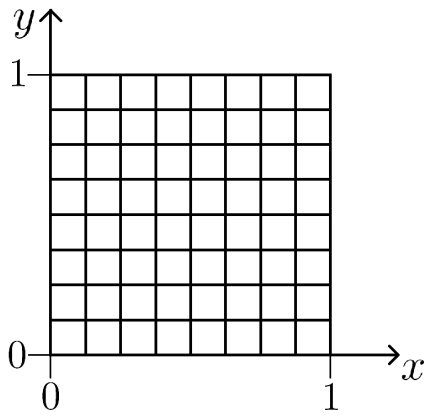


Figure 2–6: An 8×8 grid. Notice how the grid divides the domain into cells.

that has corner grid points $(i-1, j-1), (i-1, j), (i, j-1)$ and (i, j). In our case, each cell has *cell sizes* $\Delta x = \frac{1}{n}$ and $\Delta y = \frac{1}{n}$.

When using an interpolant based on a grid, we can modify algorithm 1 to get the practical version described in algorithm 2. Any method following the general structure of algorithm 2 is called semi-Lagrangian, since it combines an advection step (Lagrangian) and a projection onto an interpolation space on a grid (Eulerian). This structure is also refered to as the CIR method [5].

Algorithm 2 Solving the transport equation on a grid

· Define the initial interpolant $\bar{\phi}_0$ representing the initial set M.
for $t = \Delta t$ to ∞ **do**
· For all grid points $\vec{x}$, compute the footpoint $\mathring{x}$ at time $t - \Delta t$.
· Evaluate $\bar{\phi}(\mathring{x}, t - \Delta t)$. Copy this value to $\bar{\phi}(\vec{x}, t)$.
· Create the new interpolant $\bar{\phi}$ from the newly computed data on the grid.
· Increase t by Δt.
end for

2.2.1 Polynomial interpolation and WENO schemes

We begin by looking at interpolants in one dimension. An easy way of defining an interpolant on a 1D grid is to use linear interpolation. Assume that the function values $\phi(x_i)$ are known at every grid point. We can define an interpolant inside a grid cell by using only the information on the edges of

this cell. To achieve this, define a linear interpolant $\bar{\phi}$ inside a cell i by

$$\bar{\phi}(x) := \frac{\phi_i}{\Delta x}(x - x_{i-1}) - \frac{\phi_{i-1}}{\Delta x}(x - x_i). \tag{2.19}$$

This defines a piecewise linear interpolant on all of $[0, 1]$. Such an interpolant is shown in figure in figure 2–7. Note that the interpolant matches the true function values on the grid points, i.e. $\bar{\phi}(x_i) = \phi(x_i)\ \forall\, i$.

As was the case for Runge-Kutta methods in the previous section, we can quantify the error made when using such interpolants. This is easily done using errors bounds on Lagrange polynomials [3]. Namely, for a linear interpolant, we have

$$\bar{\phi}(x) = \phi(x) + O(\Delta x^2) \tag{2.20}$$

for any x in cell i. We say interpolant (2.19) is *locally second order accurate in space*[2] .

To have better accuracy, we can use Lagrange polynomials of higher order. For instance, the third order Lagrange polynomials is defined as the cubic polynomial interpolating 4 consecutive data points. Therefore, for a cell i, we can use points x_{i-2}, x_{i-1}, x_i and x_{i+1} to define a cubic interpolant. This is shown in figure 2–8. Such an interpolant is now locally fourth order in space, and the method can obviously be generalised to arbitrary order by using more points.

A problem already arises. We call *stencil* the set of points used to define the interpolant in a given cell. For the piecewise cubic interpolant, the stencil

[2] As opposed to the Euler method used to solve the ODE (2.15), which was locally second order *in time*.

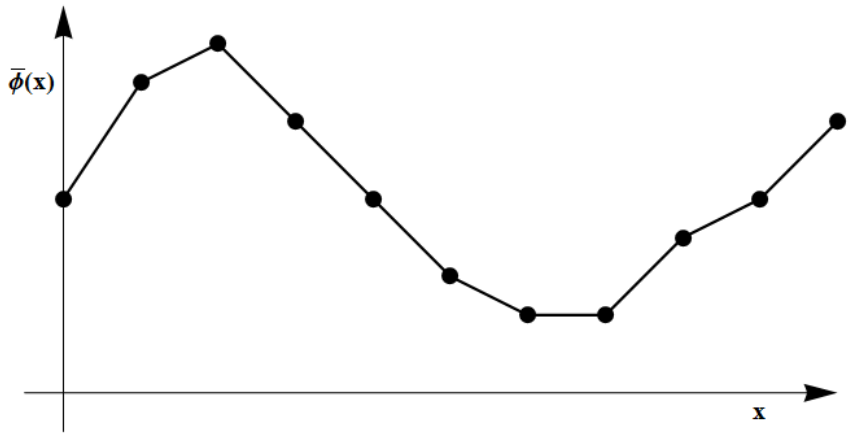


Figure 2–7: A piecewise linear interpolant. The black dots represent values of the true function ϕ, and the piecewise linear function $\bar{\phi}$ (black lines) interpolates this data.

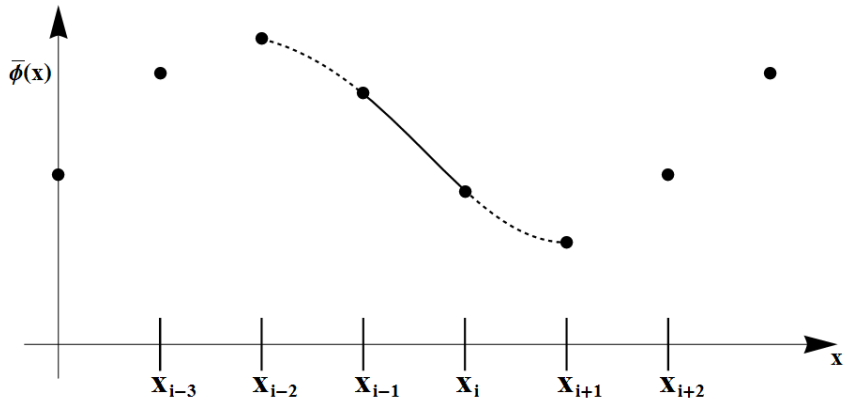


Figure 2–8: A piecewise cubic interpolant. The black dots represent values of the true function ϕ, and the piecewise cubic function $\bar{\phi}$ interpolates this data. The dashed curves are part of the same cubic, but only the solid curve is used as an interpolant.

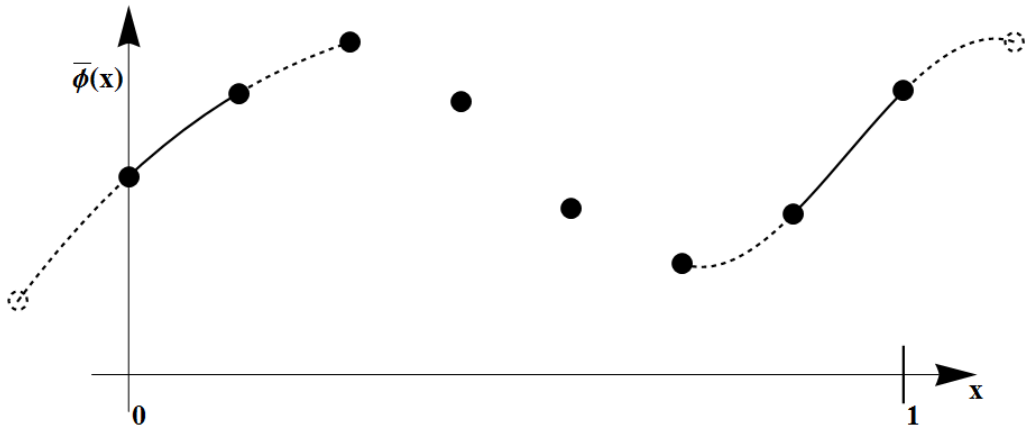


Figure 2–9: Cubic interpolant using ghost points. Points are added at each end of the $[0, 1]$ interval to allow cubic interpolation in the first and last subintervals.

goes further than the bounds of the cell. Therefore, the stencil goes outside the domain for cells 1 and n. To solve this problem, we use *ghost points*. These are artificial points added on the outside of the domain. The values of the function on ghost points is usually determined using boundary conditions of the PDE system we want to solve. An example of cubic interpolant using ghost points is shown in figure 2–9 for cells 1 and n.

One issue with polynomial interpolants is that they are bad at representing function with large slopes. Instead of having well defined jumps, polynomials of degree ≥ 2 can be highly oscillatory around non-smooth data. To overcome this problem, Weighted Essentially Non-Oscillatory (WENO) schemes are often used. We describe here the 5 point stencil version of the WENO scheme since it is the most widely used. Instead of defining the interpolants inside cells, they are defined between cell centers. Using quadratic Lagrange polynomials, we can define the interpolant in the interval $[x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}]$ in three natural ways. Namely, we can use either the points $S_1 = \{x_{i-2}, x_{i-1}, x_i\}$, $S_2 = \{x_{i-1}, x_i, x_{i+1}\}$ or $S_3 = \{x_i, x_{i+1}, x_{i+2}\}$ to define the polynomial. The idea of WENO schemes is to combine the three interpolants in a linear combination. The method

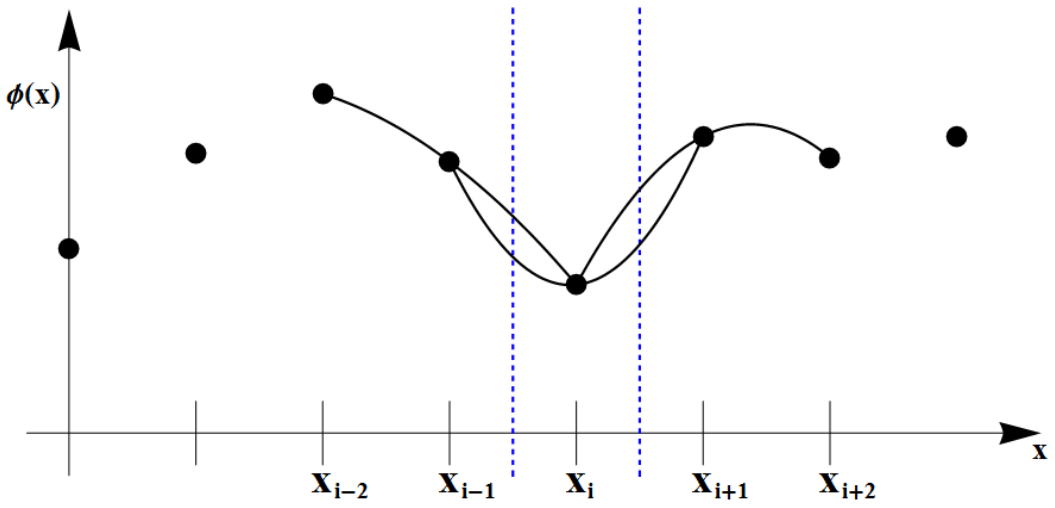


Figure 2–10: Interpolation using a WENO scheme. The interpolant in the region $[x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}]$ (bounded by the two blue dotted lines) is determined using a linear combination of three different quadratic curves.

defines a measure of smoothness for each of the three interpolants and puts more weight on smoother interpolants. Since the smoothness indicator usually depends on the function, those methods are non-linear.

In figure 2–10, we show a common situation where a sharp slope exists inside a cell $i+1$. In this case, the quadratic function using S_1 is smoother than the other two. The interpolant in between the two dotted blue lines will therefore be closer to the left quadratic curve than from the other two. Essentially, this is what we want, since the interpolants using S_2 and S_3 are more oscillatory and will tend to create more errors. All the details regarding the method can be found in [12] and [18].

The method is locally fifth order accurate for smooth functions, which is more than enough for most applications. This weighting idea gives a method that is very robust to sharp gradients in the solution. Such jumps can happen for instance when characteristic curves create shocks, which is very common and explains the popularity of WENO schemes.

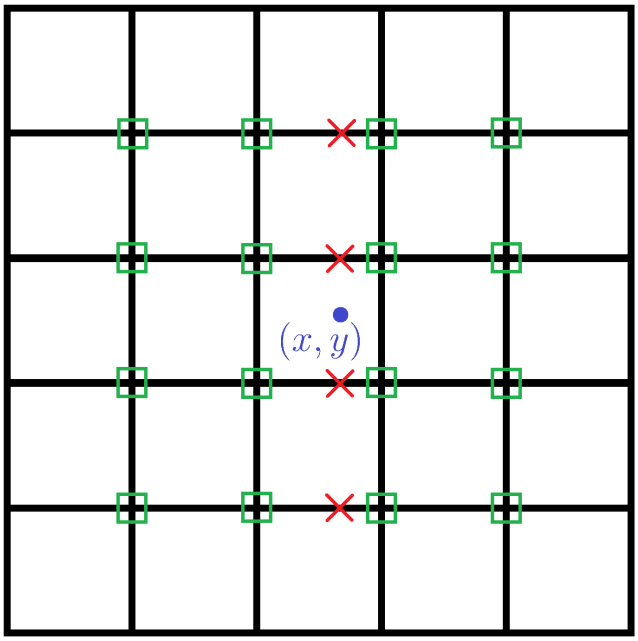


Figure 2–11: Extension of a 1D interpolant to 2D. We use the 1D interpolant in each variable sequentially. First, use an interpolation in x to evaluate the function values at the location of the red crosses. Then, using those 4 new values, we can use an interpolation in y to approximate the value at the desired (x, y) location.

WENO schemes can easily be extended to higher dimensional spaces. In fact, any 1D interpolant can be extended to higher dimension on regular grids using the following simple technique. For clarity, we explain this method using the cubic interpolant of figure 2–8. The idea is to interpolate each dimension one after the other. Figure 2–11 shows how to do this in 2D. Suppose we want to compute $\bar{\phi}(x, y)$, where the point (x, y) falls in the cell (i, j). First, use the points $(i-2, \gamma), (i-1, \gamma), (i, \gamma)$ and $(i+1, \gamma)$ (shown as green squares) to interpolate ϕ in the x direction at (x, y_γ) (shown as red crosses) for $\gamma \in \{j-2, j-1, j, j+1\}$. Then, using those four values, interpolate ϕ in the y direction at the desired location (shown as a blue dot). The exact same idea can be used for linear interpolation and WENO schemes, and can be generalized to any dimension.

2.2.2 Gradient-augmented level set method

The second interpolant we present is the main component of the Gradient-Augmented Level Set (GALS) method. It was first introduced by Nave et al. in [19], and further developed in [4] and [23]. As for WENO schemes, the method is generalizable to any order and dimension, but we present here the cubic GALS interpolant in two dimensions.

Instead of using large stencils to achieve high order interpolation, the GALS interpolant uses more data at every grid point. In addition to knowing the values of ϕ at the grid points, assume we also know the values of some of its derivatives. The collection of data on each grid point is call the *partial jet* at this grid point.

Since the 2D interpolant can be constructed using the idea of figure 2–11, we begin by presenting the interpolant in 1D. Let the jet at each grid point i consist of the function values $\phi_i := \phi(x_i)$ and its first derivative $\partial\phi_i := \partial_x\phi(x_i)$. The interpolant on grid cell i is defined as the cubic polynomial satisfying the function values and derivatives at both ends of the cell. This polynomial is called the *Hermite cubic polynomial* on this cell. To easily build the Hermite polynomial, consider the four basis functions

$$w_\alpha^v(x) = \begin{cases} f(x) & \text{if } v = 0 \text{ and } \alpha = 0 \\ f(1-x) & \text{if } v = 1 \text{ and } \alpha = 0 \\ g(x) & \text{if } v = 0 \text{ and } \alpha = 1 \\ -g(1-x) & \text{if } v = 1 \text{ and } \alpha = 1 \end{cases} \tag{2.21}$$

where $f(x) = 2x^3 - 3x^2 + 1$ and $g(x) = x(1-x)^2$. The four basis functions are shown in figure 2–12. They have the useful property that each of them has either value or derivative 1 at either $x = 0$ or $x = 1$ while having all other values

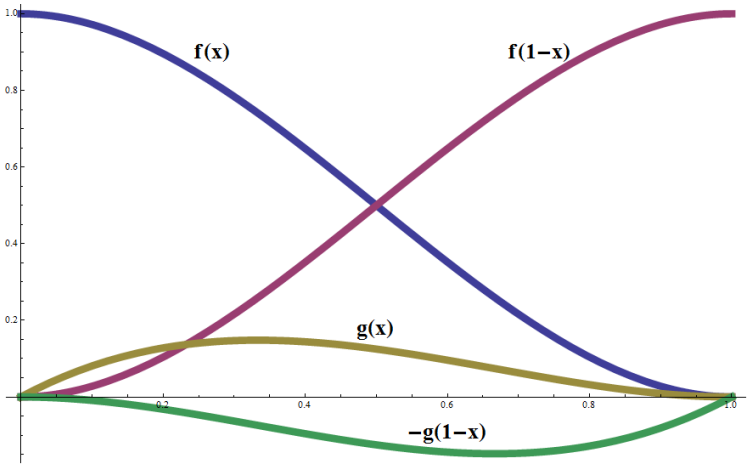


Figure 2–12: Basis functions in 1D. Each function has either value or derivative 1 at either $x = 0$ or $x = 1$ while having all other values and derivatives equal to 0.

and derivatives equal to 0. Since all four functions are cubic polynomials, the Hermite interpolant can be constructed from a linear combination of the basis functions. If $\Delta x = 1$, the interpolant on a cell i is

$$\bar{\phi}(x) = \phi_{i-1} w_0^0(x) + \phi_i w_0^1(x) + \partial\phi_{i-1} w_1^0(x) + \partial\phi_i w_1^1(x). \tag{2.22}$$

For $\Delta x \neq 1$, the basis function are scaled and the same linear combination is used.

The equivalent interpolant in 2D uses a jet composed of the function values, the x and y derivatives and the xy derivative. While the idea of interpolating dimension by dimension can be used to define the GALS interpolant in higher dimension as in figure 2–11, a generalization of the basis function idea is easier to use. We use a tensor product of 1D basis to define the 2D basis functions

$$W_{\vec{\alpha}}^{\vec{v}}(\vec{x}) = \prod_{k=1}^{2} w_{\alpha_k}^{v_k}(\vec{x}) \tag{2.23}$$

where the indices $\vec{\alpha} = \{\alpha_1, \alpha_2\}$ and $\vec{v} = \{v_1, v_2\}$ have one component for each dimension. Note that those basis functions have the similar property of having either value, derivative in x, y or in xy equal to 1 at a certain corner while all those values are 0 at the other corners. A linear combination similar to (2.22) using 16 terms can therefore be used to define a 2D bicubic interpolant in a given cell.

The main advantage of the GALS interpolant over WENO interpolants is the locality of its stencil. For the same order of accuracy (locally fourth order), the WENO interpolant uses 25 data points to define the interpolant in a given cell, while the GALS interpolant only uses the four corners of the cell. Therefore, no ghost points need to be used on the boundary, which makes the implementation easier. The price to pay is that more information has to be tracked on each grid point.

As before, the function values $\phi_{i,j}$ are transported using the transport equation (2.14)

$$\partial_t \phi + \vec{u} \cdot \nabla \phi = 0 \tag{2.24}$$

but new equations need to be used to transport the ∂_x, ∂_y and ∂_{xy} derivative. Taking the gradient of the transport equation, we get

$$\partial_t (\nabla \phi) + \nabla \left(\vec{u} \cdot (\nabla \phi) \right) = 0 \tag{2.25}$$

which is an evolution equation for $\nabla\phi$, i.e. for $\partial_x \phi$ and $\partial_y \phi$. A similar equation can be derived for the ∂_{xy} derivative. In the original paper by Nave et al. [19], these equations are used to advect the jet on the grid. However, in [23], a more convenient solution is proposed to advect derivatives. This other approach is called ϵ-*finite differences*. When we have to compute information by tracking

back a point (x, y) at time t along a characteristic, four points are tracked instead of one. Those points are $(x-\epsilon, y-\epsilon)$, $(x-\epsilon, y+\epsilon)$, $(x+\epsilon, y-\epsilon)$ and $(x+\epsilon, y+\epsilon)$ for a very small value ϵ. This results in the four footpoints

$$\mathring{x}_{-1,-1} := \mathring{x}((x-\epsilon, y-\epsilon), t-\Delta t) \tag{2.26}$$

$$\mathring{x}_{-1,+1} := \mathring{x}((x-\epsilon, y+\epsilon), t-\Delta t) \tag{2.27}$$

$$\mathring{x}_{+1,-1} := \mathring{x}((x+\epsilon, y-\epsilon), t-\Delta t) \tag{2.28}$$

$$\mathring{x}_{+1,+1} := \mathring{x}((x+\epsilon, y+\epsilon), t-\Delta t) \tag{2.29}$$

which can be combined to approximate the required jet data. These approximation are

$$\phi(x,y) \approx \frac{\mathring{x}_{-1,-1} + \mathring{x}_{-1,+1} + \mathring{x}_{+1,-1} + \mathring{x}_{+1,+1}}{4} \tag{2.30}$$

$$\partial_x \phi(x,y) \approx \frac{-\mathring{x}_{-1,-1} - \mathring{x}_{-1,+1} + \mathring{x}_{+1,-1} + \mathring{x}_{+1,+1}}{4\epsilon} \tag{2.31}$$

$$\partial_y \phi(x,y) \approx \frac{-\mathring{x}_{-1,-1} + \mathring{x}_{-1,+1} - \mathring{x}_{+1,-1} + \mathring{x}_{+1,+1}}{4\epsilon} \tag{2.32}$$

$$\partial_{xy} \phi(x,y) \approx \frac{\mathring{x}_{-1,-1} - \mathring{x}_{-1,+1} - \mathring{x}_{+1,-1} + \mathring{x}_{+1,+1}}{4\epsilon^2}. \tag{2.33}$$

By taking ϵ small enough, these approximations can be made as accurate as required to maintain the order of accuracy of the scheme. The ϵ-finite differences approach therefore allows to use the same transport equation (2.14) as before to compute all the derivatives required by the GALS interpolant.

2.3 Improving level set schemes

Since the level set methods are very general, they sometimes have to be modified to better suit specific problems. This section presents some possible modifications that either facilitate implementation, speed up calculations or improve stability.

2.3.1 Boundary conditions

In some situations, the behavior of the solution at the boundary of the computational domain is of prime importance. Physical constraints usually impose this behavior through boundary conditions. Those conditions also depend on the vector field $\vec{u}$ at the boundary.

Depending on the situation, the characteristic curves may cross the boundary of the domain, and therefore exit the region on which the level set function is defined. This is a problem, since if a footpoint falls outside the domain, we cannot evaluate the level set function at this location.

Figure 2–13 shows three possible situations that can occur. The left figure shows a flow where the vector field is zero on the boundary of the computational domain. This often happens in fluid simulation and is called a *no slip boundary condition*. This situation is not problematic because no characteristic can cross the boundary, and footpoints always fall inside the domain. The middle figure shows a vector field that is periodic across the boundary. In such a situation, if a footpoints falls outside the domain, it can be brough back to the other side. For instance, if a footpoints exists to the right of the domain, it is equivalent to it entering through the left, so this situation is also easy to handle.

Finally, the right panel of figure 2–13 shows a vector field that exits the domain without any particular boundary condition. This is the most problematic situation since a footpoint exiting the domain cannot be brought back inside it in a natural way. Still, if the set we are tracking is far away from the boundary, we can solve this problem by changing the level set function. Consider the situation where the initial set is a circle of radius 0.25 centered at $(0.5, 0.5)$ in the domain $[0, 1] \times [0, 1]$. We want to transport this circle in the

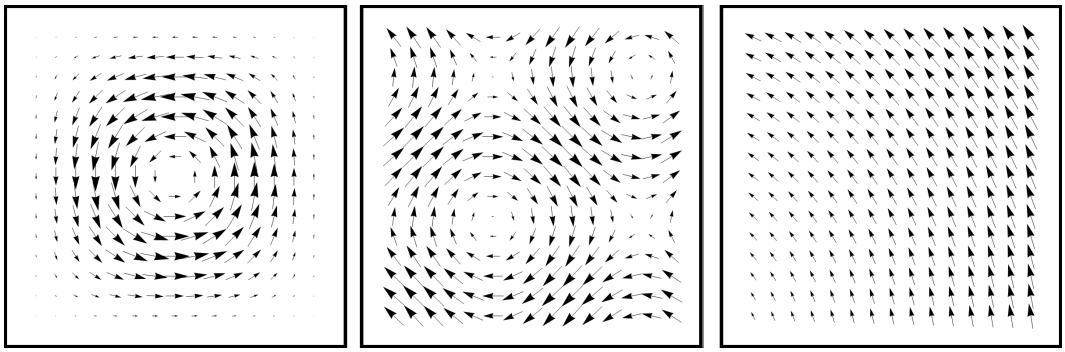


Figure 2–13: Three examples of behavior around the boundary. The left figure shows a no slip condition, the middle figure shows a periodic boundary condition, and the right figure shows a case where the flow enters and exists the domain without any particular boundary condition.

vector field of the right image in figure 2–13. The situation is shown in the left part of figure 2–14. This figure also shows two possible level set functions to represent the circle. The left one is the standard level set function as defined by equation (2.8). The right one is a level set function that is constant around the boundary of the domain. For the left one, if a footpoint exits the domain, we have to use some extrapolation to evaluate the function. In this case it is possible since the level set function is simple, but for a more deformed function, this can be hard to do. For the right level set, the extrapolation is trivial since the function is constant around the boundary. This trick can be used whenever the set we are evolving does not come close to the boundary.

2.3.2 Narrow band methods

In section 2.3.1, we saw that the level set function can be modified away from the zero level set without changing the represented set. Therefore, only the region of the level set function laying around the zero level set matters. This was originally a big drawback of level set methods since most of the computed level set function is not used.

Figure 2–14: Level set modification for a flow crossing boundaries. The left figure shows the case of a circle transported in a vector field that enters and exits the domain. The middle figure shows the level set function we would usually use, and the right figure shows a modified level set function that is constant around the boundary, allowing it to be evaluated outside the domain through trivial extrapolation.

To solve this problem, new methods were developed that only compute the level set function in a band around the manifold being transported. For this reason, these methods are generally refered to as *narrow band* methods, and a good example is found in [1]. Before performing an advection step of the level set function, these methods first identify the cells that are located within a given distance β away from the zero level set. This distance is usually of the order of 3 cell widths. In these cells, the level set function is evaluated as usual by tracking characteristics. Note that since the set moves in time, the cells where the function is to be evaluated have to be redefined at every step.

To evaluate which cells are part of the band, the level set function can be used directly. If this function is built using definition (2.8), it is a signed distance function to the set, so the band is initially made of grid points where $|\phi_{i,j}| \leq \beta$.

Narrow band methods significantly reduce the computational cost of level set methods. For the regular level set method, the computational cost per time step is $C_1(n+1)^2$, where C_1 is the time taken to compute the new value

of the level set function for one grid point. Asymptotically, for large values of n, we say the methods has a computational cost of $O(n^2)$. For a narrow band level set method, this cost becomes $C_1 b(n, \beta)$, where $b(n, \beta)$ is the number of grid points in the narrow band. For fine enough grids, i.e. for large enough n, $b(n, \beta)$ behaves like $O(n)$, so the method is one order faster than usual level set methods. In many industrial applications, the grids used have very high resolution to ensure high accuracy of the solutions, so narrow band methods are greatly beneficial in such situations.

2.3.3 Reinitialization

As was emphasized at the beginning of this chapter, it is desirable to have a level set function with unit slopes. But even if this condition is met for the initial level set function, it might be violated as the function is deformed by the vector field. For this reason, a *reinitialization step* is sometimes used. The purpose of this step is to replace the current level set function by another one having the same zero level set, but with $||\nabla\phi|| = 1$ almost everywhere.

The most natural way of doing it is to take the current level set function and identify its zero level set M. Then, the level set function is replaced by the solution of the system

$$\begin{cases} ||\nabla\psi|| = 1 \\ \psi = 0 \quad \text{on } M. \end{cases} \tag{2.34}$$

This system is called the *Eikonal equation*. The intuition is somewhat violated as (2.34) uses explicitely the set M, which breaks the idea of having all the information enclosed in the level set function. But numerous methods can be used to solve (2.34) efficiently. See for instance [24].

Another popular way of achieving $||\nabla\phi|| = 1$ was proposed by Sussman et al. in [27]. Starting with a function ψ_0, they use the equation

$$\partial_\tau \psi = sign(\psi_0)(1 - |\nabla\psi|). \tag{2.35}$$

A steady state solution of (2.35) is a function having $||\nabla\psi|| = 1$ with the same zero level set as ψ_0. This solution can be computed using an interative process. In practive, only a few iterations are necessary for the level set function to guarantee a signed distance function, at least near $\phi = 0$.

Even though they are not essential, reinitialization steps are useful to ensure stability in situations where the level set function can become greatly distorted. However, for the numerical tests presented in this thesis, no reinitialization step was used. We will see in the following chapters that using the GALS framework does not require to maintain $||\phi|| = 1$.

CHAPTER 3
Particle and hybrid methods

In this chapter, we present a completely different way to represent and transport sets in a vector field. Instead of using an implicit representation of sets as in chapter 2, we use an explicit description. We will then merge the explicit approach with level set methods to create hybrid methods.

We call *Lagrangian particles* points that are advected passively in the vector field $\vec{u}$. Recall from chapter 2 that such particles follow characteristic curves. In other words, if a point $\hat{x}(t)$ is a Lagrangian particle, it satisfies the ODE

$$\partial_t \hat{x}(t) = \vec{u}(\hat{x}(t), t). \tag{3.1}$$

Consider an initial set $M(0)$. One way of transporting it in a vector field $\vec{u}$ is to transport each individual point of M using equation (3.1). The transported set $M(t)$ is then the collection of all transported initial points. Of course, this cannot be done in practice since the sets we evolve are usually not finite. Still, we can take a finite sample of points from the set, advect those points forward along characteristics and then try to recover the whole transformed set from the sample.

3.1 Recovering sets from particles.

Consider the 2D case. A natural way of using particles would be to define curves explicitely. For instance, instead of defining a circle by

$$x^2 + y^2 - 1 = 0 \tag{3.2}$$

we could define it by

$$\{(\cos(\theta), \sin(\theta)) \,|\theta \in [0, 2\pi[\} \,. \tag{3.3}$$

A numerical approximation of this set could then be to take the points

$$\left\{ \left(\cos\left(\frac{2\pi i}{m}\right), \sin\left(\frac{2\pi i}{m}\right)\right) \middle| \, i = \{0, ..., m-1\} \right\} \tag{3.4}$$

for a given integer m and join all consecutive points by a line segment. But this method has several problems. First, recall that we want to differentiate the interior of the set from its exterior. With the explicit representation, there is no trivial way of doing this.

Figure 3–1 shows two other problems that occur with the naive particle representation. The left image shows the initial set, two circles centered horizontally in the domain $[0, 1]^2$. The numerical solution is shown in black and the true solution is shown in red. The middle image shows the result after some time in a vector field that stretches the circles into ellipses. Because we didn't use enough particles, the points get far from each other at the left and right edges of the ellipses, and the approximated solution is not accurate. Of course, taking more particles initially would reduce this error, but in general we do now know in advance how many particles should be used.

The figure on the right shows the same initial condition, but now the flow causes the two circles to cross. This is a problematic situation, since points that were initially on the boundary are not anymore. At this stage, we would have to delete some particles and completely redefine the connections between the remaining particles to have the correct representation of the set, which is not an easy task. This problem becomes even more complicated in 3D, where there is no natural ordering of the particles.

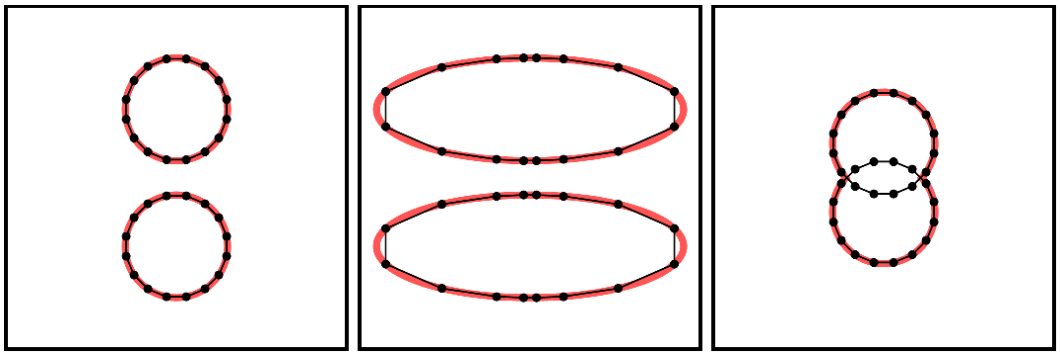


Figure 3–1: Problems occuring with the explicit particle representation. The initial set (red) sampled with particles (black dots) is joined by line segments (black). The middle image shows a flow that deformed the circles into ellipses. We see that the line segments are not a good approximation of the true set at the left and right bound of the ellipses. The right image shows a case where the two initial circles collapse. The particle representation is also problematic here since some particles are not on the set anymore.

The following sections present two methods found in the litterature that can be used to correctly recover the set from a particle sample.

3.1.1 Moving least squares method

The moving least square (MLS) method was first introduced by Lancaster and Salkauskas in [16]. The description we give here is based on [22]. The method comes in various forms, but we present the basics of the MLS methods.

The MLS method uses a level set function to represent the set, but it does it in a very different way than what was presented in the previous chapter. We start by taking particles in the domain. We call the set of those particles the *particle cloud* and identify each particle by an index $p \in \mathcal{P}$. The position of those particles will be noted $\hat{x}_p$. The initial positions of those particles can be taken randomly or strategically. For instance, we might want to concentrate particles around the zero level set since this is where the information is most important, like in the left image in figure 3–3. Then on each particle we record the signed distance to the initial set. Note that this is equivalent to building

the level set function with definition (2.8) and then recording the value of the function at the particle locations. For this reason, we denote the values on the particles by $\hat{\phi}_p = \phi_0(\hat{x}_p)$. We will will often refer to $\hat{\phi}_p$ as the *data* on the particles.

We recall the very important property of the transport equation (2.14) that the function value on a given particle never changes over time. This was stated before by equation (2.11). We therefore only need to compute $\hat{x}_p$, and leave $\hat{\phi}_p$ constant. We then want to recover the level set function from the particle cloud so that we can take its zero level set and recover the transported set $M(t)$.

The MLS method reconstructs an approximation of the function as a polynomial. For a given polynomial basis $\{b_1(\vec{x}), ..., b_k(\vec{x})\}$, any polynomial $P(\vec{x})$ in the corresponding polynomial space can be written as

$$P(\vec{x}) = \sum_{j=1}^{k} a_j b_j(\vec{x}) = \vec{a} \cdot \vec{b}(\vec{x}) \tag{3.5}$$

where $\vec{a} = \{a_1, ..., a_k\}$ is a set of coefficients. The method uses the polynomial that best fits the particle data around a point $\vec{x}$ to define the polynomial interpolant at this point $\vec{x}$. For a given set of particles $\hat{x}_p$ with data $\hat{\phi}_p$, $p \in \mathcal{P}$, the method minimizes the functional

$$E_{\hat{x}} := \sum_{p \in \mathcal{P}} w\left(\vec{x}, \hat{x}_p\right) \left(P(\hat{x}_p) - \hat{\phi}_p\right)^2 \tag{3.6}$$

where $w\left(\vec{x}, \hat{x}_p\right)$ is a weight function. It is important to notice that since the functional $E_{\hat{x}}$ depends on $\vec{x}$, we get a different polynomial interpolant for different evaluation points. To minimize the functional, the problem is rewritten in terms of the weights $\vec{a}$. The minimization problem then reduces to solving a linear system.

Many weight functions can be chosen, but they usually have the form

$$w\left(\vec{x}, \hat{x}_p\right) = \Theta(||\vec{x} - \hat{x}_p||) \tag{3.7}$$

for some positive function $\Theta(d)$. This function determines the behavior of the whole method. In general, we want the polynomial interpolant to be defined using mostly the information located around the point $\vec{x}$. For this reason, the original paper [16] suggests using

$$\Theta(d) = \exp\left(-\alpha d^2\right) \tag{3.8}$$

for some parameter α. Using this weighting function, all particles are taken into account when computing the interpolant. But for computational reasons, it might be preferable to compute the minimization using only particles that fall in a small neighbourhood of the point $\vec{x}$. To achieve this, a popular choice is to take

$$\Theta(d) = \begin{cases} \exp(-(d^2 - d_0^2)^{-2}) & \text{if } d \leq d_0 \\ 0 & \text{if } d > d_0 \end{cases} \tag{3.9}$$

for some predetermined maximal distance d_0, which gives a weighting function with compact support. The two weight functions are represented in figure 3–2 for $\alpha = 1$ and $d_0 = 1$, with a normalization so that they both have unit area. Recall that the minimization is done by solving a linear system, and the size of this system depends on the number of particles we use. In a context where we have thousands of particles throughout the domain, having a compactly supported weight function is highly profitable.

The MLS method, like most particle based methods, has the disadvantage that if not enough particles are used, the interpolant might be undetermined in some regions. This can happen for instance if we use the compactly supported

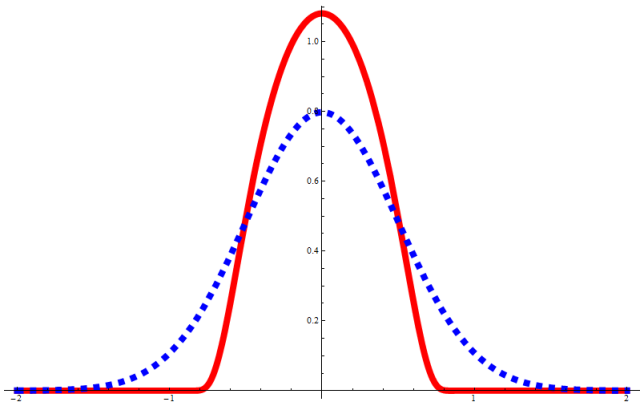


Figure 3–2: Two different weight functions Θ that can be used in the MLS method. The dashed blue line is function (3.8) and the full red line is function (3.9). Both functions put emphasis on particles closer to the evaluation point, but the red weighting has a compact support, reducing greatly computational costs.

weighting function (3.9) and no particle is present in a d_0 disk around $\vec{x}$. Making sure that this situation does not happen can be complicated, since the particle density changes as the set evolves. Figure 3–3 shows a situation where particles are originally placed in a band around the initial set, allowing to determine the level set function around its zero level set. However, after some time, the particles concentrate in certain areas and avoid other regions. The right panel of figure 3–3 shows that the level set function cannot be evaluated at a point $\vec{x}$ since no particle falls in its d_0 neighbourhood. Various procedures of particle management have been developed to add and delete particles as time evolves to make sure that the particle density is appropriate at all times, at the expense of complicating the algorithm.

3.1.2 Circle envelope

The second method we present was developed by Enright et al. in [6]. We describe a simplified version in 2D for clarity.

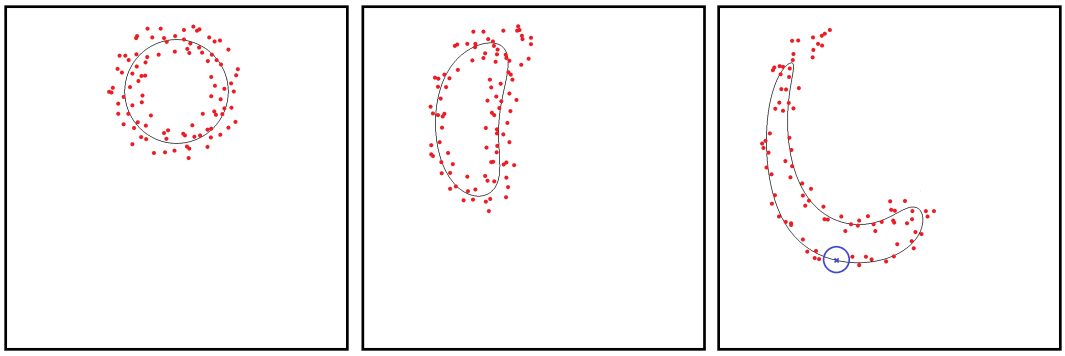


Figure 3–3: Evolution of particle density for a problematic case. Particles (red dots) are initially placed in a band around the set (black), which is enough to reconstruct the level function around its zero level set. As time evolves, the particle density changes, and in the right image, no particle falls in the d_0 neighbourhood (blue circle) of a point $\vec{x}$ (blue cross), so the function cannot be evaluated there.

As in section 3.1.1, we begin with a particle cloud. These particles are initiated in a band around the initial curve. Around each particle, we put a circle tangent to the curve and centered at the particle. This radius is simply given by the distance from the particle to the curve. The envelope of the circles is then used to represent the curve. The left image in figure 3–4 shows a curve (red) represented by the envelope (blue) of many circles (black). Once again, if not enough points are used, the curve is poorly represented. The right image in figure 3–4 shows the same curve and its representation using more particles, which gives a better result. As the particle density goes to infinity, the curve will be exactly represented.

This method suffers from the same problem as the MLS method of section 3.1.1, that is, if after some time the particle density in a region becomes too low, the method will not give good results. Therefore, particle management also has to be used for this method. Another problem is that when particles follow characteristic curves, the radii of their circles has to be changed. The transport equation only states that the level set function values stay constant on particle,

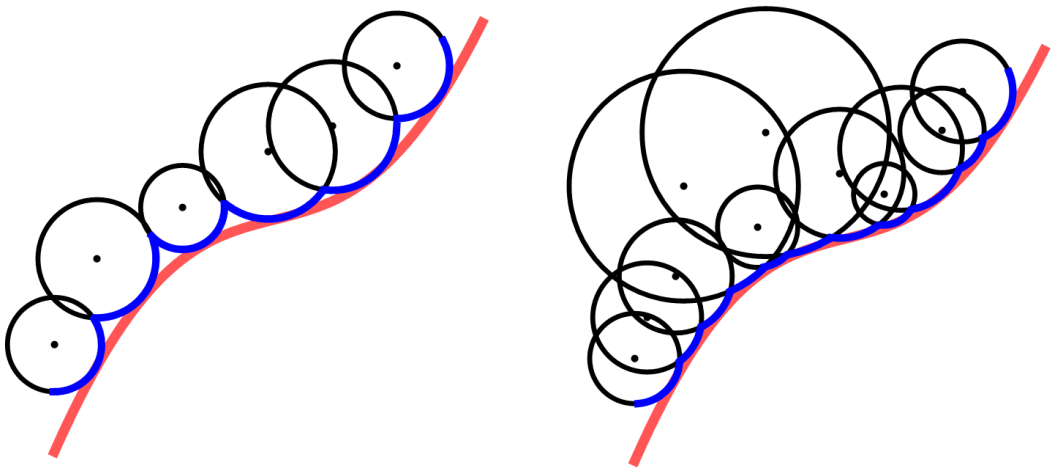


Figure 3–4: Curve represented by the envelope of circles. Particles (black dots) are placed around the original curve (red). Circles (black) tangent to the curve are placed around particles, and their envelope (blue) is used to represent the curve. We see that using more particules increases the quality of the representation.

but it is not true that the distance from a particle to the set remains constant. As a result, the particles need to follow different trajectories so that they remain at their respective distance from the set. To achieve this, the circles are pushed in the direction normal to the set until they are tangent again. Even though the method requires a lot of particles and some complicated and potentially costly particle management steps, it is relatively efficient since advecting particles if very cheap (solving ODEs) compared to computing level functions (solving a PDE).

In the original paper [6], the method is presented as a hybrid level set method. The idea is to use a regular level set method, and then to use the particle representation in regions where the level set method fails. To detect those situations, we compare the particle position to the sign of the level set function. If a particle changes sign, it indicates the level set function is not accurate anymore. Once again, this step involves a lot of details and we refer to [6] for in depth explanations.

3.2 Hybrid methods

We concluded the last section by proposing a method in which particles can be used in conjunction with level set functions in a hybrid manner. One advantage of particle methods over interpolants defined on a grid is that the advection of particles is much easier to compute and we can use a high order solver to compute the particle positions accurately. When using semi-Lagrangian schemes, the error is usually more important since errors from the interpolation step add to the error made when tracking footpoints [19]. This makes particle clouds highly desirable information, and incorporating it in semi-Lagrangian schemes can increase accuracy.

In this section, we present two novel ways to incorporate particles in the GALS formulation presented in section 2.2.2.

3.2.1 Least squares minimization

One way of using particles in the GALS environment is to use them to modify the Hermite interpolant. To do this, we add a modification step in algorithm 2. The result is shown in algorithm 3.

Algorithm 3 Solving the transport equation on a grid with particles

· Define the initial interpolant $\bar{\phi}_0$ representing the initial set M.
for $t = \Delta t$ to ∞ **do**
  · Modify the interpolant using the particle information.
  · For all grid points $\vec{x}$, compute the footpoint $\mathring{x}$ at time $t - \Delta t$.
  · Evaluate $\bar{\phi}(\mathring{x}, t - \Delta t)$. Copy this value to $\bar{\phi}(\vec{x}, t)$.
  · Create the new interpolant $\bar{\phi}$ from the newly computed data on the grid.
  · Increase t by Δt.
end for

We start with a particle cloud as in section 3.1.1. On each of those particles, we store the function value $\hat{\phi}(\hat{x}_p)$ for each particle $p \in \mathcal{P}$. Particles are transported passively in the vector field by equation (3.1).

Since the particles are a trusted sample of the true function ϕ, the idea is to modify the GALS interpolant $\bar{\phi}$ in a cell so that it matches more closely the values on the particles in that cell. More precisely, we want to minimize $\left(\bar{\phi}(\hat{x}_p) - \hat{\phi}_p\right)^2$. To achieve this, we replace the interpolant $\bar{\phi}$ in a cell $\mathcal{C}$ by a modified interpolant defined by

$$\bar{\phi}^{\text{mod}} = \underset{\phi \in \mathcal{H}}{\text{argmin}} \left(\frac{\alpha}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \left(\phi(\hat{x}_p) - \hat{\phi}_p\right)^2 + \frac{\beta}{\Delta x^d} \int_C \left(\phi(\vec{x}) - \bar{\phi}(\vec{x})\right)^2 d\vec{x} \right) \quad (3.10)$$

where α and β are weights, $\mathcal{H}$ is the set of all Hermite interpolants on cell $\mathcal{C}$ and $|\mathcal{P}|$ is the number of particles in that cell. The first term in (3.10) approaches the interpolant from the particles, and the second term restricts the modified interpolant from being too far from the initial interpolant. Note also the factor $\frac{1}{\Delta x^d}$ that makes both terms unitless. The following lemma solves the minimisation problem (3.10).

Lemma 1. *If the original interpolant $\bar{\phi}$ is represented by a linear combination of k basis functions w_i by $\bar{\phi}(\vec{x}) = \bar{a} \cdot (w_1(\vec{x}), ..., w_k(\vec{x}))^T$, $\bar{a} \in \mathbb{R}^k$, then the solution of (3.10) is $\bar{\phi}^{mod}(\vec{x}) = \bar{a}^{mod} \cdot (w_1(\vec{x}), ..., w_k(\vec{x}))^T$ where*

$$\bar{a}^{mod} = \left(\frac{\alpha}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \hat{w}_p \hat{w}_p^T + \frac{\beta}{\Delta x^d} W \right)^{-1} \left(\frac{\alpha}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \hat{w}_p \hat{\phi}_p + \frac{\beta}{\Delta x^d} W \bar{a} \right) \quad (3.11)$$

and where

$$\hat{w}_p := (w_1(\hat{x}_p), ..., w_k(\hat{x}_p))^T$$

$$\bar{a} := (\bar{a}_1, ..., \bar{a}_k)^T$$

$$\bar{a}^{mod} := \left(\bar{a}_1^{mod}, ..., \bar{a}_k^{mod}\right)^T$$

$$\mathbb{R}^{k \times k} \ni W := \left\{ \int_C w_i(\vec{x})\ w_j(\vec{x})\ d\vec{x} \right\}_{\substack{i=1:k \\ j=1:k}}$$

Proof. We directly have from (3.10) that

$$\bar{\phi}^{\text{mod}} = \underset{\phi\in\mathcal{H}}{\text{argmin}} \quad \frac{\alpha}{|\mathcal{P}|}\sum_{p\in\mathcal{P}}\left(\phi(\hat{x}_p)^2 - 2\phi(\hat{x}_p)\hat{\phi}_p + \hat{\phi}_p^2\right) + \frac{\beta}{\Delta x^d}\int_C \left(\phi(\vec{x})^2 - 2\phi(\vec{x})\bar{\phi}(\vec{x}) + \bar{\phi}(\vec{x})^2\right) d\vec{x} \quad (3.12)$$

$$\bar{\phi}^{\text{mod}} = \underset{\phi\in\mathcal{H}}{\text{argmin}} \quad \frac{\alpha}{|\mathcal{P}|}\sum_{p\in\mathcal{P}}\left(\phi(\hat{x}_p)^2 - 2\phi(\hat{x}_p)\hat{\phi}_p\right) + \frac{\beta}{\Delta x^d}\int_C \left(\phi(\vec{x})^2 - 2\phi(\vec{x})\bar{\phi}(\vec{x})\right) d\vec{x} \quad (3.13)$$

since $\hat{\phi}_p^2$ and $\int_{\mathcal{C}} \bar{\phi}(\vec{x})^2 d\vec{x}$ do not depend on ϕ. Writting the minimisation in term of the weights $\bar{a}$, we get

$$\bar{a}^{\text{mod}} = \underset{a}{\text{argmin}} \quad \frac{\alpha}{|\mathcal{P}|}\sum_{p\in\mathcal{P}}\left(\left(\sum_{i=1}^k a_i w_i(\hat{x}_p)\right)^2 - 2\left(\sum_{i=1}^k a_i w_i(\hat{x}_p)\right)\hat{\phi}_p\right) + \frac{\beta}{\Delta x^d}\int_C\left(\left(\sum_{i=1}^k a_i w_i(\vec{x})\right)^2 - 2\left(\sum_{i=1}^k a_i w_i(\vec{x})\right)\left(\sum_{i=1}^k \bar{a}_i w_i(\vec{x})\right)\right) d\vec{x} \quad (3.14)$$

$$= \underset{a}{\text{argmin}} \quad \frac{\alpha}{|\mathcal{P}|}\sum_{p\in\mathcal{P}}\left(\left(\sum_{i=1}^k\sum_{j=1}^k a_i a_j w_i(\hat{x}_p) w_j(\hat{x}_p)\right) - 2\left(\sum_{i=1}^k a_i w_i(\hat{x}_p)\right)\hat{\phi}_p\right) + \frac{\beta}{\Delta x^d}\int_C\left(\left(\sum_{i=1}^k\sum_{j=1}^k a_i a_j w_i(\vec{x}) w_j(\vec{x})\right) - 2\left(\sum_{i=1}^k\sum_{j=1}^k a_i \bar{a}_j w_i(\vec{x}) w_j(\vec{x})\right)\right) d\vec{x}$$

$$= \underset{a}{\text{argmin}} \quad \frac{\alpha}{|\mathcal{P}|}\sum_{p\in\mathcal{P}}\left(\left(\sum_{i=1}^k\sum_{j=1}^k a_i a_j w_i(\hat{x}_p) w_j(\hat{x}_p)\right) - 2\left(\sum_{i=1}^k a_i w_i(\hat{x}_p)\right)\hat{\phi}_p\right) + \frac{\beta}{\Delta x^d}\left(\left(\sum_{i=1}^k\sum_{j=1}^k a_i a_j \int_C w_i(\vec{x}) w_j(\vec{x})\, d\vec{x}\right) - 2\left(\sum_{i=1}^k\sum_{j=1}^k a_i \bar{a}_j \int_C w_i(\vec{x}) w_j(\vec{x})\, d\vec{x}\right)\right)$$

$$= \underset{a}{\text{argmin}} \quad \frac{\alpha}{|\mathcal{P}|}\sum_{p\in\mathcal{P}}\left(a^T\hat{w}_p\hat{w}_p^T a - 2a^T\hat{w}_p\hat{\phi}_p\right) + \frac{\beta}{\Delta x^d}\left(a^T W a - 2a^T W\bar{a}\right) \quad (3.15)$$

$$= \underset{a}{\text{argmin}} \quad a^T \left[\frac{\alpha}{|\mathcal{P}|} \left(\sum_{p \in \mathcal{P}} \hat{w}_p \hat{w}_p^T \right) + \frac{\beta}{\Delta x^d} W \right] a + a^T \left[-2 \frac{\alpha}{|\mathcal{P}|} \left(\sum_{p \in \mathcal{P}} \hat{w}_p \hat{\phi}_p \right) - 2 \frac{\beta}{\Delta x^d} W \bar{a} \right] \quad (3.16)$$

Note that by construction the matrix $\left[\frac{\alpha}{|\mathcal{P}|} \left(\sum_{p \in \mathcal{P}} \hat{w}_p \hat{w}_p^T \right) + \frac{\beta}{\Delta x^d} W \right]$ is positive definite since, from (3.14), we see that $a^T \left[\frac{\alpha}{|\mathcal{P}|} \left(\sum_{p \in \mathcal{P}} \hat{w}_p \hat{w}_p^T \right) + \frac{\beta}{\Delta x^d} W \right] a$ represents a sum of squares. It is thus invertible, and the quadratic minimization (3.16) is solved as usual by

$$\bar{a}^{\text{mod}} = -\frac{1}{2} \left[\frac{\alpha}{|\mathcal{P}|} \left(\sum_{p \in \mathcal{P}} \hat{w}_p \hat{w}_p^T \right) + \frac{\beta}{\Delta x^d} W \right]^{-1} \left[-2 \frac{\alpha}{|\mathcal{P}|} \left(\sum_{p \in \mathcal{P}} \hat{w}_p \hat{\phi}_p \right) - 2 \frac{\beta}{\Delta x^d} W \bar{a} \right]$$

$$= \left[\frac{\alpha}{|\mathcal{P}|} \left(\sum_{p \in \mathcal{P}} \hat{w}_p \hat{w}_p^T \right) + \frac{\beta}{\Delta x^d} W \right]^{-1} \left[\frac{\alpha}{|\mathcal{P}|} \left(\sum_{p \in \mathcal{P}} \hat{w}_p \hat{\phi}_p \right) + \frac{\beta}{\Delta x^d} W \bar{a} \right] \quad (3.17)$$

□

Note that if no particle fall in a given cell, the modified interpolant does not change. We therefore do not have to worry about particle management during a simulation as was the case with the methods presented in sections 3.1.1 and 3.1.2.

We now show some results using the hybrid least squares method.

Swirl test

This test uses a 2D divergence-free deformation field defined by

$$u(x, y, t) = \cos\left(\frac{\pi t}{8}\right) \sin^2(\pi x) \sin(2\pi y)$$
$$v(x, y, t) = -\cos\left(\frac{\pi t}{8}\right) \sin^2(\pi y) \sin(2\pi x)$$

on the domain $[0,1] \times [0,1]$. This is the classical *vortex in a box* test found in [17]. The initial curve is a circle of radius 0.15 centered at $(0.5; 0.75)$. This velocity induces swirling of the curve until it reaches its maximum elongation, and then returns it back to its original position in a period of time $[16q, 16(q+1)]$ for $q \in \mathbb{N}$, which will be denoted here as a *cycle*. We used a 64×64 grid with time step $\Delta t = \Delta x$. We place the particles in a band of width $6\Delta x$ around the original interface, with a density of 1 particle per cell, resulting in a total of 361 particles. The weights used are $\alpha = 1$ and $\beta = 15$. Figure 3–5 shows the results for $t = 0, 8$ and 16, which corresponds to $0, 0.5$ and 1 cycles of the swirl. The same test was also done using a higher density of particles of 10 particles per cell for a total of 3610 particles, and results are shown in figure 3–6.

The results show an important feature of the hybrid method. At $t = 8$, the surface is at its most stretched position, and we see that an important part of the surface is thinner than the grid size. The gradient-augmented level set method does a good job of keeping track of these subgrid structures with its cubic interpolant, but inevitably loses information over time. In contrast, the hybrid method is able to recover these lost structures of the surface. Since the particles are advected independently of the curve, they don't suffer at all from the subgrid structures or the interpolation. They are thus capable of retaining the information in the thin regions, and this information is transfered back to the interpolant. This shows that the interpolation error of the gradient-augmented level set method is partially compensated by the use of the particle information. The results at $t = 16$ show that having this supplementary information drastically improves the representation of the curve in time.

The results of figure 3–5 show that even with a few particles, the tracking of the interface is improved. But with a higher density, as in figure 3–6, the results are improved. First of all, when comparing the two cases for 0.5 cycles (middle figures), we see that the thin tail of the surface gains a lot of precision from the additional particles. Also, after one complete cycle (bottom figures), we see that the surface with few particles has some wrinkles in its upper left region. These wrinkles completely disappear when using a higher density of particles.

The hybrid method is even more beneficial for longer periods of time. As seen in figure 3–7, the GALS method loses information about the interface everytime it goes through a large stretch, and the method cannot retrieve the lost information. The results of figure 3–7 show that for 9.5 and 10 cycles, the interface advected by the GALS method progressively loses accuracy, while the hybrid method keeps the interface intact.

Zalesak's circle test

We consider in this section the 2D rigid rotation of a slotted disk around the point $(0.5; 0.5)$ in the domain $[0, 1] \times [0, 1]$, as proposed by [28]. The initial surface is a circle of radius 0.15 centered at $(0.5; 0.75)$ with a slot of width 0.05 and length 0.25. The rotation is induced by the velocity field $\vec{u} = (u, v)$ where

$$u(x, y, t) = \frac{2\pi}{628}\left(\frac{1}{2} - y\right)$$
$$v(x, y, t) = \frac{2\pi}{628}\left(x - \frac{1}{2}\right).$$

Tests were done on a 64×64 grid and a time step $\Delta t = 1$, so that the circle completes a full revolution in 628 steps. We used a density of 10 particles per cell spread uniformly on a disk of radius one cell larger than the one in the initial surface. Results are shown in figure 3–8 for 0, 5, 10 and 100 revolutions.

The results show how the particles help to preserve the shape of the slotted disk, especially in regions with sharp corners. The interpolation step using cubic polynomials tends to smooth those regions, but the particle representation doesn't lose these structures over time. Modifying the interpolant thus allows to recover the correct shape of the surface. The results after 100 rotations are especially remarkable, since the original method loses all trace of the slot, while the solution from the hybrid method is almost perfect.

3D deformation field

There is no fundamental difference between the method in two and three dimensions. We test here the advection of a surface under a divergence free 3D deformation field given by

$$
\begin{aligned}
u(x,y,z,t) &= \cos\left(\frac{\pi t}{2}\right) 2\sin^2(\pi x)\sin(2\pi y)\sin(2\pi z) \\
v(x,y,z,t) &= -\cos\left(\frac{\pi t}{2}\right) \sin(2\pi x)\sin^2(\pi y)\sin(2\pi z) \\
w(x,y,z,t) &= -\cos\left(\frac{\pi t}{2}\right) \sin(2\pi x)\sin(2\pi y)\sin^2(\pi z)
\end{aligned}
$$

This velocity field is found in [17] and combines deformations in the x-y and the x-z plane. We perform this test on a $50 \times 50 \times 50$ grid. The initial surface is a sphere of radius 0.15 centered at $(0.35, 0.35, 0.35)$. We compare our hybrid method with the GALS method. Particles are spread uniformly between the spheres of radius $0.15 - 2\Delta x$ and $0.15 + 2\Delta x$ centered at $(0.35, 0.35, 0.35)$. Results are shown in figure 3–9.

The results show improvements qualitatively similar to those from the 2D swirl test of section 3.2.1. After one cycle, we see that the hybrid method made a much better job of preserving the surface. Most of the errors come from the stretching of the surface in the middle region of figures c) and d). This region

is very thin and the subgrid structure is partialy lost by the original GALS method. The hybrid method is once again able to recover the information from the particles and correct the damage done in this region.

Stability issues

Although the results from the hybrid least squares method are promising, the method suffers from stability issues. To see this, we look at a 1D interpolant. The left side of figure 3–10 shows a 1D interpolant (blue) in a cell with one particle (red). We apply the modification step on this interpolant 100 times (green curves). The 3 first results are labeled.

We see that the modification step is not a projection, meaning that applying the step twice does not give the same result as applying it once. This is a problem, since the modified interpolant can then become very different than the initial interpolant. Still, the iterations converge, but they converge to an interpolant that matches the particles exactly without taking into account the initial interpolant. The speed of this convergence depends on the weights α and β in (3.10). This is a problem in a situation such as the one shown in the right part of figure 3–10. The initial interpolant is shown in blue, and 4 particles (red) are in the cell. The limit solution matches all particles exactly, but this results in a highly oscillatory solution that is far away from the original interpolant. This in turns results in unstable simulations if the weight α is too large compared to β. Prescribing weights that prevent instabilities is difficult and no clear criterion has been found yet.

3.2.2 Taylor extrapolation from particles

In order to address some of the shortcomings of the original hybrid idea presented above, we attempt to use particles in a different way. The second

original method we present uses particles in a way similar to [6]. Instead of using particles to modify the GALS interpolant as in the previous section, we use particles to create a new interpolant, and then choose between the GALS and the new interpolant. A selection step is added to the original algorithm 2 to get algorithm 4.

Algorithm 4 Solving the transport equation with taylor expansions from particles

· Define the initial interpolant $\bar{\phi}_0$ representing the initial set M.
for $t = \Delta t$ to ∞ **do**
· For all grid points $\vec{x}$, compute the footpoint $\mathring{x}$ at time $t - \Delta t$.
· Create the Taylor interpolant from the particle cloud.
· Choose between the Taylor interpolant and the GALS interpolant.
· Evaluate $\bar{\phi}(\mathring{x}, t - \Delta t)$. Copy this value to $\bar{\phi}(\vec{x}, t)$.
· Create the new interpolant $\bar{\phi}$ from the newly computed data on the grid.
· Increase t by Δt.
end for

The *Taylor interpolant* is described here in 1D for simplicity. We begin with a particle cloud on which we store not only the function values $\hat{\phi}_p = \phi(\hat{x}_p)$, but also the derivatives up to second order. It is important to note that even though the function values $\hat{\phi}_p$ do not change over time, the derivatives do and we need a differential equation to compute the evolution of those derivatives. This is easily done by taking the corresponding derivatives of equation (3.1) and more details about solving these new equations can be found in [19]. We will assume that we know the value of the derivatives $\widehat{\partial_x \phi}_p$ and $\widehat{\partial_{xx} \phi}_p$ at all time, as well as $\hat{\phi}_p$. As was mentioned before, those values are computed by solving simple ODEs and can therefore be computed very accurately, so we will suppose they are exact in what follows.

With this extra data, we can build around each particle a Taylor extrapolation of ϕ and use it as an interpolant. The Taylor extrapolation around a

particle $\hat{x}_p$ is

$$T_p(x) = \hat{\phi}_p + \widehat{\partial_x \phi}_p (x - \hat{x}_p) + \widehat{\partial_{xx} \phi}_p \frac{(x - \hat{x}_p)^2}{2} + O((x - \hat{x}_p)^3). \tag{3.18}$$

When we want to approximate ϕ at a given footpoint, we have to choose between the GALS interpolant $\bar{\phi}$ and all the Taylor interpolants T_p. In general, the Taylor interpolants are only usable if the footpoint falls close enough to the particle. We therefore use the Taylor interpolant if the footpoint is close to a given particle, and otherwise we use the GALS interpolant.

We can use the error of the Taylor expansion to have an exact decision criterion. Using the higher order terms in the Taylor expansion, we know that

$$|\phi(x) - T_p(x)| = \left| \partial_{xxx} \phi(\xi) \frac{(x - \hat{x}_p)^3}{6} \right| \tag{3.19}$$

for some ξ between x and $\hat{x}_p$. It might be difficult to evaluate this error, but assume we at least have an upper bound $C_p \geq |\partial_{xxx} \phi(\xi)| \quad \forall\, \xi \in [x, \hat{x}_p]$.

We also need to quantify the error made by the GALS interpolant so we can choose which interpolant is best. Looking at $\phi(x) - \bar{\phi}(x)$ as a function itself, we can Taylor expand it around a particle $\hat{x}_p$ to get

$$|\phi(x) - \bar{\phi}(x)| \tag{3.20}$$

$$\begin{aligned} &\approx \Big| \left[\phi(\hat{x}_p) - \bar{\phi}(\hat{x}_p) \right] + \left[\partial_x \phi(\hat{x}_p) - \partial_x \bar{\phi}(\hat{x}_p) \right] (x - \hat{x}_p) \\ &\quad + \left[\partial_{xx} \phi(\hat{x}_p) - \partial_{xx} \bar{\phi}(\hat{x}_p) \right] \frac{(x - \hat{x}_p)^2}{2} + \left[\partial_{xxx} \phi(\hat{x}_p) - \partial_{xxx} \bar{\phi}(\hat{x}_p) \right] \frac{(x - \hat{x}_p)^3}{6} \Big| \end{aligned}$$

$$\begin{aligned} &= \Big| \left[\hat{\phi}_p - \bar{\phi}(\hat{x}_p) \right] + \left[\widehat{\partial_x \phi}_p - \partial_x \bar{\phi}(\hat{x}_p) \right] (x - \hat{x}_p) \\ &\quad + \left[\widehat{\partial_{xx} \phi}_p - \partial_{xx} \bar{\phi}(\hat{x}_p) \right] \frac{(x - \hat{x}_p)^2}{2} + \left[\partial_{xxx} \phi(\hat{x}_p) - \partial_{xxx} \bar{\phi}(\hat{x}_p) \right] \frac{(x - \hat{x}_p)^3}{6} \Big|. \end{aligned}$$

Define

$$E_p(x) := \left| \left[\hat{\phi}_p - \bar{\phi}(\hat{x}_p) \right] + \left[\widehat{\partial_x \phi}_p - \partial_x \bar{\phi}(\hat{x}_p) \right] (x - \hat{x}_p) + \left[\widehat{\partial_{xx} \phi}_p - \partial_{xx} \bar{\phi}(\hat{x}_p) \right] \frac{(x - \hat{x}_p)^2}{2} \right|. \quad (3.21)$$

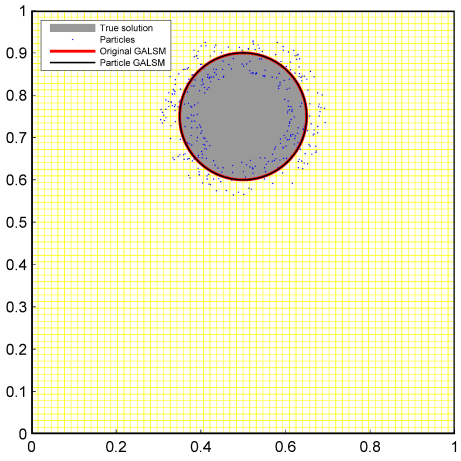
Using the same bound C_p as above, and noticing that $\partial_{xxx}\bar{\phi}$ is constant since the GALS interpolant is a cubic polynomial, we can use the triangle inequality to get

$$E(x) - \left| \left[C_p - \partial_{xxx} \bar{\phi}(\hat{x}_p) \right] \frac{(x - \hat{x}_p)^3}{6} \right| \leq \left| \phi(x) - \bar{\phi}(x) \right| \leq E(x) + \left| \left[C_p - \partial_{xxx} \bar{\phi}(\hat{x}_p) \right] \frac{(x - \hat{x}_p)^3}{6} \right| \quad (3.22)$$

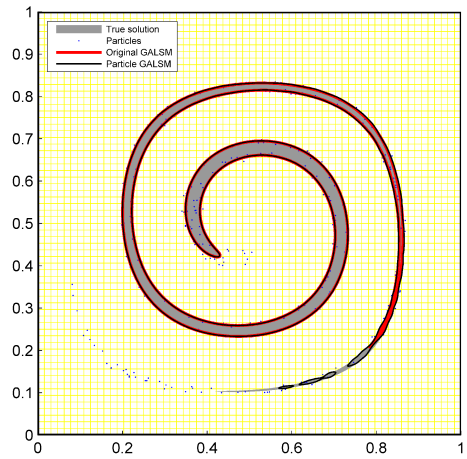
for values of x where $E(x) - \left| \left[\partial_{xxx}\phi(\xi) - \partial_{xxx}\bar{\phi}(\xi) \right] \frac{(x-\hat{x}_p)^3}{6} \right| \geq 0$. Note that this last inequality is true at least in small neighbourhood of $\hat{x}_p$, unless $E(\hat{x}_p) = 0$, but in this case the particle and the GALS interpolant agree so any interpolant we pick will give the same result.

Figure 3–11 helps to put all the pieces together. It shows the different error curves described above. The postion of the particle $\hat{x}_p$ is shown by the dashed line. Curve 1 (black) shows the error of the Taylor interpolant depending on the position of x around the particle. Curves 2 (red) and 3 (blue) show the two bounds given in (3.22) for the error of the GALS interpolant.

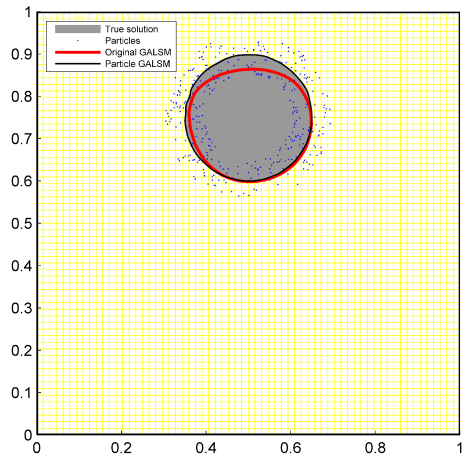
The criterion for choosing the interpolant is then very simple : Let S be the largest interval containing x_p such that the error of the Taylor interpolant is smaller than the lowest bound of the GALS error $\forall x \in S$. We choose the Taylor interpolant if $x \in S$ and use the GALS interpolant otherwise.



(a) Initial condition.

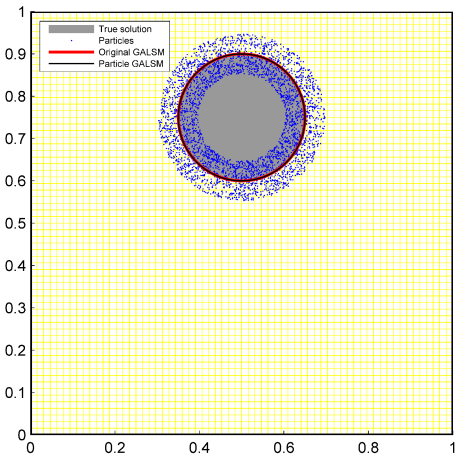


(b) 0.5 cycles.

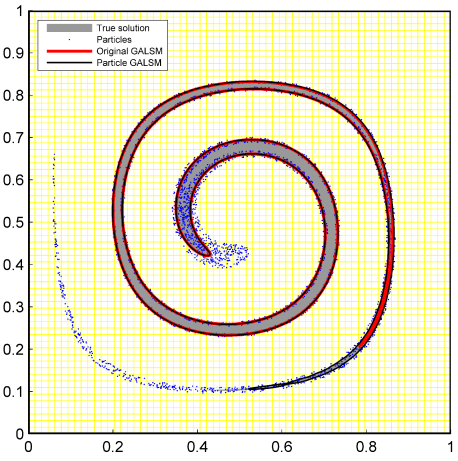


(c) 1 cycles.

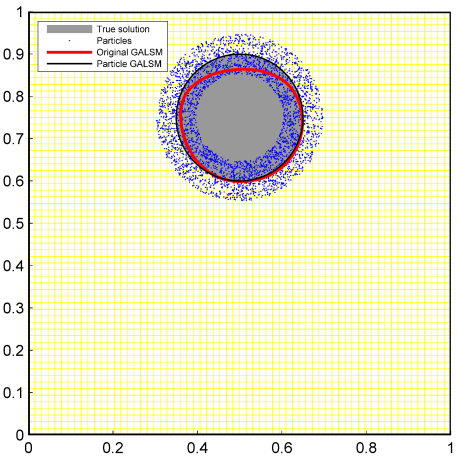
Figure 3–5: Hybrid least squares method. Swirl test with low particle density for 0, 0.5 and 1 cycle.



(a) Initial condition.

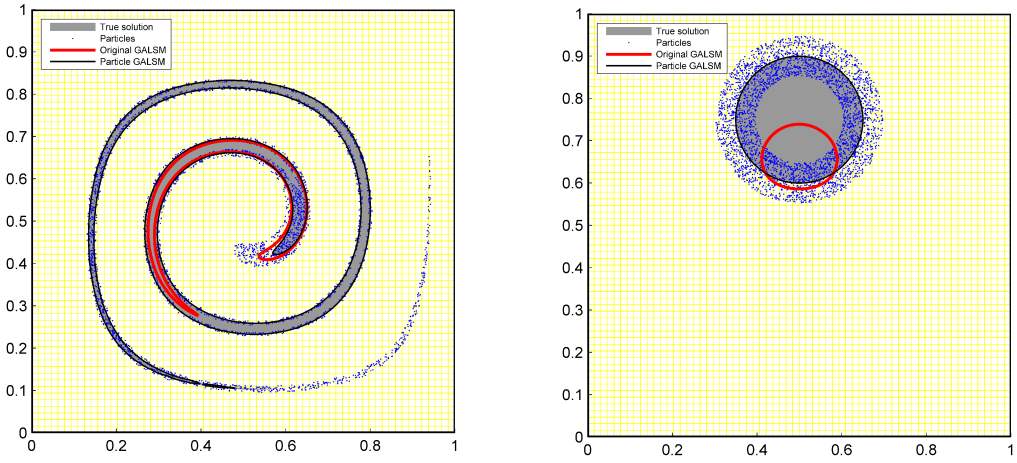


(b) 0.5 cycle.



(c) 1 cycle.

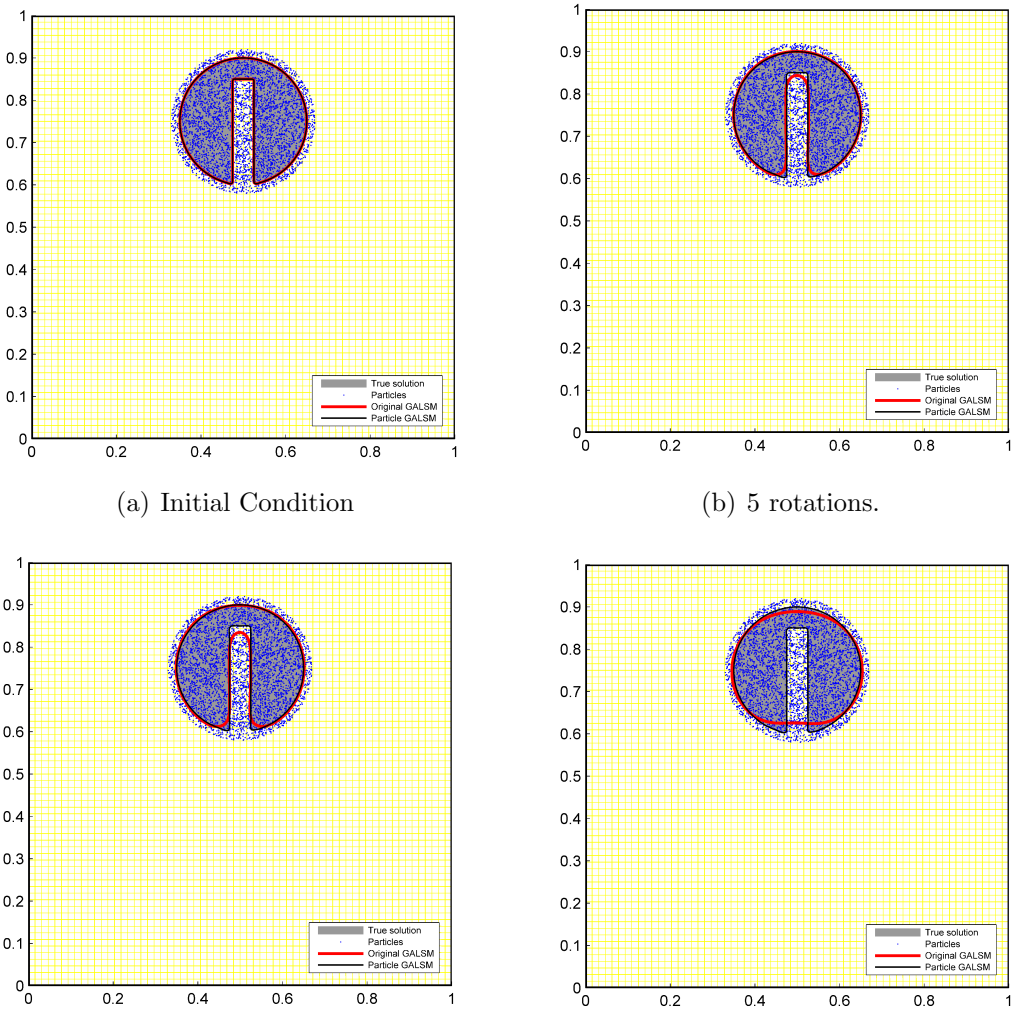
Figure 3–6: Hybrid least squares method. Swirl test with high particle density for 0, 0.5 and 1 cycle.



(a) 9.5 cycles.

(b) 10 cycles.

Figure 3–7: Hybrid least squares : Long term result for the swirl test



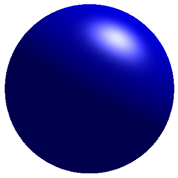
(a) Initial Condition

(b) 5 rotations.

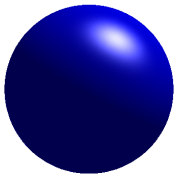
(c) 10 rotations.

(d) 100 rotations.

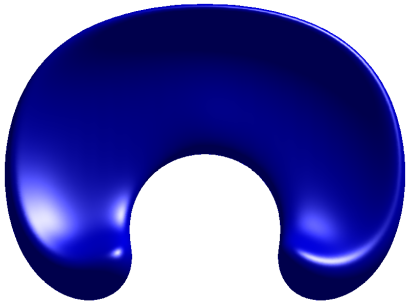
Figure 3–8: Hybrid least squares : results for the Zalesak's disk test.



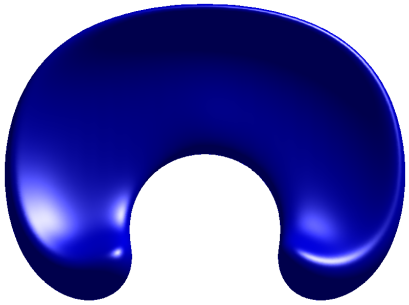
(a) Initial condition.



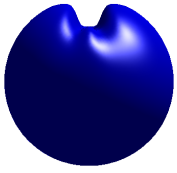
(b) Initial condition.



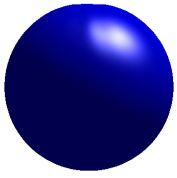
(c) 0.5 GALS method.



(d) 0.5 cycle hybrid method.



(e) 1 cycle GALS method.



(f) 1 cycle hybrid method.

Figure 3–9: 3D deformation test at 0, 0.5 and 1 cycle. Original GALS method on the left, hybrid least squares method on the right.

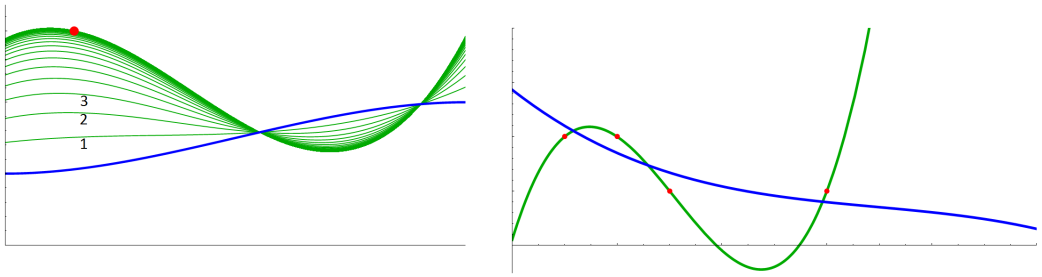


Figure 3–10: Convergence of the least squares modification. Left: The modification step is not a projection, and applying it iteratively converges to a solution that matches the particle exactly without taking the initial interpolant into account. Right: The limit solution can be highly oscillatory and far away from the initial interpolant.

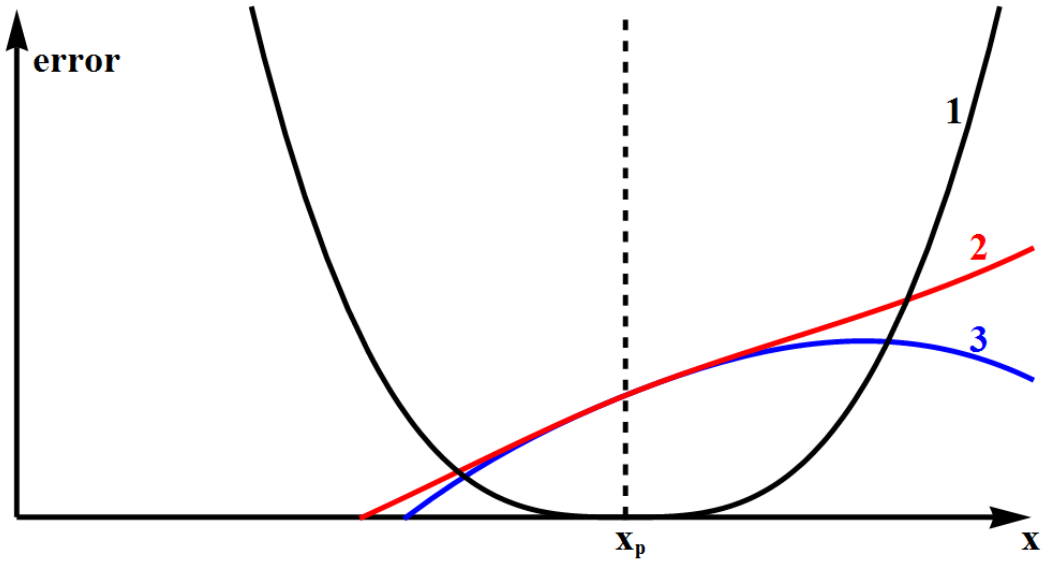


Figure 3–11: Error from Taylor and GALS interpolants. Curve 1 (black) shows the error from the GALS interpolant. Curves 2 and 3 show the error bounds of the Taylor interpolant from equation (3.22)

CHAPTER 4
Set transport through diffeomorphisms

In this chapter, we present the *characteristic mapping* (CM) method, a novel grid-based method to solve the transport equation (2.14). The main idea of the method is that instead of transporting a set directly, we compute the transformation map of the whole domain. Unlike the methods presented in chapters 2 and 3 which only allowed to advect smooth manifolds, the characteristic mapping method can transport *any* set in a vector field.

This idea has already been used for small elastic deformations [13], but our method applies to more general contexts. The evolution of a mapping of the domain was also used in work by Pons [21], but their mapping is used to advect parametric information about the set and not the set itself. In our case, the advection of the set is completely encapsulated in the transformation map. We also use a remapping idea that is related to the one presented in [15].

Some aspects of the method come from the following consideration. In many applications, the vector field $\vec{u}$ retains relatively low frequency structures over time while the deformed set develops very fine and high frequency structures. Figure 4–1 shows such a situation, where the vector field is constant and smooth across the domain, but the initial circle gets deformed in a very fine structure. This discrepancy between the behaviors of the vector field and the set inspired the idea of using different grid sizes to represent each of them.

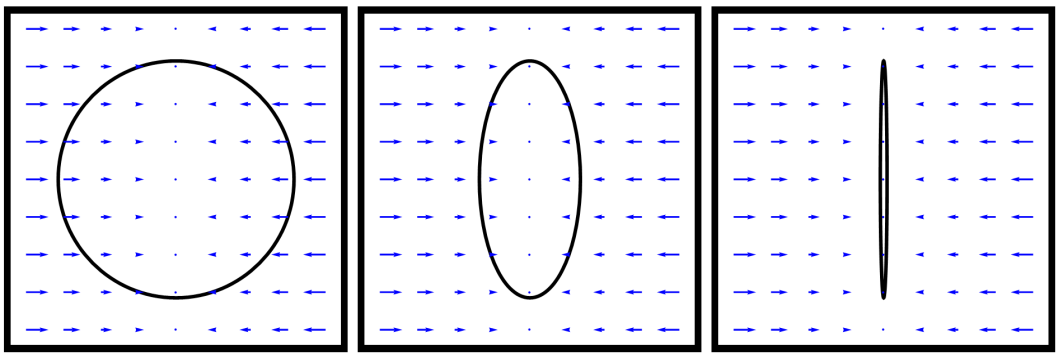


Figure 4–1: Smooth vector field inducing sharp structures. Even if the vector field is constant and smooth, it deforms the initial circle into a very sharp structure.

4.1 Mathematical formulation

We begin by presenting the formulation used by the characteristic mapping method on a regular Cartesian grid in a domain $U \subset \mathbb{R}^d$. Given a vector field $\vec{u}(\vec{x}, t)$, we want to evolve a set defined by a set function $S_0(\vec{x})$. A special case of this is when S_0 is a level set function representing a smooth manifold as defined by (2.8), but in general we do not impose any restriction on the set function S_0.

Assume we have a diffeomorphism $\vec{\chi}_0(\vec{x}, t) : U \to U$ such that for any initial point $\vec{x}(t = 0) \in U$ being evolved under the vector field $\vec{u}$, we have

$$\vec{\chi}_0(\vec{x}(t), t) = \vec{x}(0). \tag{4.1}$$

Such a diffeomorphism is depicted in figure 4–2. Under $\vec{u}$, the set initially defined by S_0 is evolved to a final time T as $S(\vec{x}, T) = S_0(\vec{\chi}_0(\vec{x}, T))$.

Note that this evolution can be decomposed into an arbitrary number of steps. We can subdivide the interval $[0, T]$ into n subintervals $[\tau_0, \tau_1], \ldots, [\tau_{n-1}, \tau_n]$ where $\tau_0 = 0$ and $\tau_n = T$. Define n diffeomorphisms $\vec{\chi}_i(\vec{x}, t), i \in \{1, \ldots, n\}$ such that $\vec{\chi}_i(\vec{x}(t), t) = \vec{x}(\tau_{i-1})\ \forall t \in [\tau_{i-1}, \tau_i]$. By doing so, we have at the final

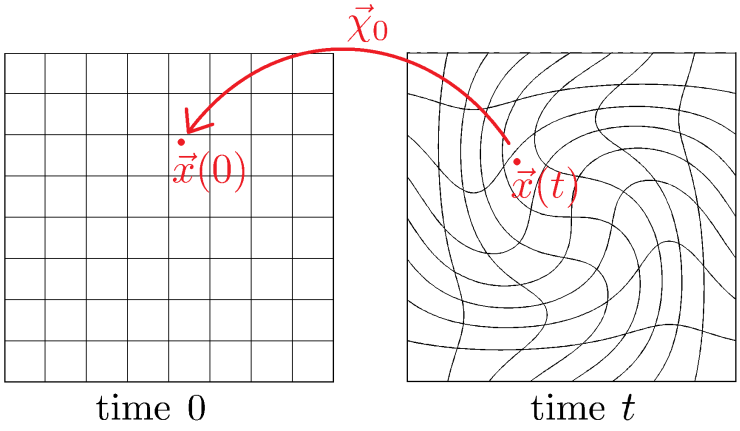


Figure 4–2: The diffeomorphism $\vec{\chi}_0$ takes a points $\vec{x}$ at time t and returns its original position at time $t = 0$.

time T that

$$\vec{\chi}_0(\vec{x}, T) = \vec{\chi}_1(\vec{\chi}_2(\ldots \vec{\chi}_n(\vec{x}, \tau_n)\ldots, \tau_2), \tau_1). \tag{4.2}$$

The $\vec{\chi}_i$ and the τ_i need to be defined properly. First, we turn our attention to the mappings $\vec{\chi}_i$. Each mapping $\vec{\chi}_i(\vec{x}, t)$ is formally defined as the solution of the advection problem

$$\frac{\partial \vec{\chi}_i}{\partial t} + \vec{u} \cdot \nabla \vec{\chi}_i = 0 \qquad \forall\, t \geq \tau_{i-1},\ x \in U \tag{4.3a}$$

$$\vec{\chi}_i(\vec{x}, \tau_{i-1}) = \vec{x}. \tag{4.3b}$$

To solve these equations numerically, we use the gradient augmented level set method presented in section 2.2.2. The equations (4.3a) are solved on grids having N_c cells per dimension (N_c^d total cells).

For the first mapping $i = 1$, the starting time $\tau_0 = 0$ is known, so we can start by solving (4.3a) for $\vec{\chi}_1$. Note that since the initial condition (4.3b) is linear, it is exactly represented by the Hermite cubic basis. However, this property may be violated after some time under the flow defined in (4.3a). This depends on specific characteristics of the velocity field $\vec{u}$. To control the

induced representation errors, we want to be able to detect situations where the interpolation error of $\vec{\chi}_1$ becomes larger than a predefined tolerance $\mathcal{E}_1$. To evaluate this error, we use Lagrangian particles $\hat{x}_p(t), p \in \{1, \ldots, m\}$, initially distributed uniformly in U at positions $\hat{x}_p^0$. Those particles are independently evolved under $\vec{u}$ by solving the set of ODEs

$$\frac{\partial \hat{x}_p(t)}{\partial t} = \hat{x}_p(t) \qquad \forall\, t \geq 0 \quad \forall\, p \in \{1, \ldots, m\} \tag{4.4a}$$

$$\hat{x}_p(0) = \hat{x}_p^0. \tag{4.4b}$$

Note once again that (4.4a) can be accurately solved by using a sufficiently high order Runge-Kutta solver. In practice, we use the same integration scheme as for the GALS method, i.e. RK3. Note that when solving (4.3a), each step of the GALS method produces two errors, one due to the approximate time integration and the other due to interpolation :

$$\text{GALS error per step} = \underbrace{O(\triangle t^4)}_{\text{time integration}} + \underbrace{O(\triangle x^4)}_{\text{Hermite interpolation}} . \tag{4.5}$$

By using the same $O(\triangle t^4)$ time integration scheme to solve (4.4a) for particles $\hat{x}_p$, a measure of the accumulated interpolation error is given by

$$M_1(\vec{\chi}_1(\vec{x}, t), t) := \max_p ||\vec{\chi}_1(\hat{x}_p(t)) - \hat{x}_p^0||. \tag{4.6}$$

Equiped with the error measure M_1, we define τ_1 as the first time at which the evolution of $\vec{\chi}_1$ induces a representation error greater than $\mathcal{E}_1$, that is

$$M_1(\vec{\chi}_1(\vec{x}, \tau_1 - \triangle t), \tau_1 - \triangle t) \leq \mathcal{E}_1 < M_1(\vec{\chi}_1(\vec{x}, \tau_1), \tau_1). \tag{4.7}$$

At this time, we stop evolving $\vec{\chi}_1$. Notice that choosing $\mathcal{E}_1$ sufficiently small ensures that $\vec{\chi}_1$ is still well represented by the Hermite cubic interpolant at $t = \tau_1$. Consequently, we may oversample $\vec{\chi}_1$ onto a finer grid with N_f cells

per dimension (N_f^d total cells) by naturally using the Hermite cubic structure of the GALS. We call this finer representation $\vec{\chi}_0$, because it will gradually become the global transformation defined in (4.1).

For all subsequent steps, we construct the mapping $\vec{\chi}_0$ recursively. When τ_{i-1} is determined, (4.3a) defines the next mapping $\vec{\chi}_i$. We solve for $\vec{\chi}_i$ and define τ_i as the first time when $M_1(\vec{\chi}_i(\vec{x}, \tau_i), \tau_i) > \mathcal{E}_1$. At this time, we stop computing the evolution of $\vec{\chi}_i$, we update $\vec{\chi}_0$ by

$$\vec{\chi}_0(\vec{x}) \leftarrow \vec{\chi}_0(\vec{\chi}_i(\vec{x}, \tau_i)). \tag{4.8}$$

and we continue with the evolution of $\vec{\chi}_{i+1}$. Every update of $\vec{\chi}_0$, either by the first oversampling of $\vec{\chi}_1$ or by (4.8), is called a *remapping step*. Finally, at time τ_n, $\vec{\chi}_0$ is the mapping defined in (4.2) and we have $S(\vec{x}, T) = S_0(\vec{\chi}_0(\vec{x}, T))$, as wanted.

Remark 1. Taking N_f large enough ensures that the composition of mappings in (4.8) stays well represented in $\vec{\chi}_0$ by the Hermite cubic basis for all time.

Remark 2. At any intermediate time $t \in [\tau_{i-1}, \tau_i]$, the evolved set can be evaluated by

$$S(\vec{x}, t) = S_0(\vec{\chi}_0(\vec{\chi}_i(\vec{x}, t - \tau_{i-1}))). \tag{4.9}$$

Remark 3. With the progressive construction of $\vec{\chi}_0$, the final time T doesn't need to be known in advance. The evolution can continue for as long as desired.

Remark 4. To ensure the particles are always spread uniformly over the domain U, we reinitialize their position at every remapping step, that is $\hat{x}_p(\tau_{i-1}) = \hat{x}_p^0$.

In the next section, we discuss in more detail the implementation of the CM method.

4.2 Numerical implementation

In this section, we detail the numerical implementation of the method described in section 4.1. The first subsection describes the use of a dynamic grid for $\vec{\chi}_0$ and the second summarizes the CM method in pseudo-code for ease of implementation.

4.2.1 Dynamic grid resolution

Given that the grid on which $\vec{\chi}_0$ lives is fine enough, all the details of the transformation induced by the vector field $\vec{u}$ can be well represented by the CM method up to an arbitrary time. However, representation problems during a remapping step can arise because in equation (4.8), the right hand side $\vec{\chi_0} \circ \vec{\chi_i}$ is a function that can be inadequately represented by the Hermite interpolant of the GALS, even though $\vec{\chi}_0$ and $\vec{\chi_i}$ are accurately represented on the fine grid. To ensure that the grid is always fine enough to represent $\vec{\chi_0} \circ \vec{\chi_i}$, we can dynamically modify the resolution of the N_f^d grid used for $\vec{\chi}_0$.

To do so, we do not immediately update $\vec{\chi}_0$ by (4.8) when a remapping time τ_i is reached. Instead, we compute the remapping in a temporary Hermite cubic interpolant $\vec{\chi_0}^{\text{temp}}$ living on the same N_f^d grid. This is done by evaluating

$$\vec{\chi_0}^{\text{temp}}(\vec{x}) = \vec{\chi}_0(\vec{\chi}_i(\vec{x}, \tau_i - \tau_{i-1})). \tag{4.10}$$

Now, for an arbitrary point $\vec{x}$ that does not live on the N_f^d grid, $|\vec{\chi_0}^{\text{temp}}(\vec{x}) - \vec{\chi}_0(\vec{\chi}_i(\vec{x}, \tau_i - \tau_{i-1}))|$ will in general be non-zero. Therefore, the measure

$$M_2(\vec{\chi_0}^{\text{temp}}) := ||\vec{\chi_0}^{\text{temp}} - \vec{\chi}_0 \circ \vec{\chi}_i||_\infty \tag{4.11}$$

will be large if the interpolant $\vec{\chi}_0^{\text{temp}}$ does not represent $\vec{\chi_0} \circ \vec{\chi}_i$ accurately enough. Consequently, we decide to redefine $\vec{\chi}_0$ on a $(2N_f)^d$ grid if $M_2(\vec{\chi}_0^{\text{temp}}) \geq \mathcal{E}_2$, where $\mathcal{E}_2$ is a predefined tolerance.

Conversely, we want to take advantage of situations where the fine grid could be coarsened. To detect such situations, we define another interpolant $\vec{\chi}_0^{\text{coarse}}$ on a coarser $\left(\frac{N_f}{2}\right)^d$ grid and compute (4.10) on this new grid. If for this coarser mapping we have $M_2(\vec{\chi}_0^{\text{coarse}}) \leq \mathcal{E}_2$, it means that the coarser mapping is sufficiently accurate to represent $\vec{\chi_0} \circ \vec{\chi}_i$. In this case, we redefine $\vec{\chi}_0$ on a $\left(\frac{N_f}{2}\right)^d$ grid.

Once the size of the fine grid for $\vec{\chi}_0$ is set, we apply the remapping step defined in (4.8) as before. Using a dynamic grid has two main advantages. By refining the grid when needed, we ensure that the transformation is always well represented, and by coarsening it when possible, we reduce the computational time required to do a remapping step. Note that the redefinition of $\vec{\chi}_0$ on a different grid is easily performed at low cost due to the Hermite cubic interpolant structure.

4.2.2 Pseudo-code algorithm

We summarize here the CM algorithm with the modification discussed in section 4.2.1. Additionally, the following two minor adjustments are done for convenience and efficiency.

First, we start by defining $\vec{\chi}_0$ on the N_f^d grid and initialize it to $\vec{\chi}_0(\vec{x}) = \vec{x}$. By doing so, the first remapping can be treated just like all the following remappings by equation equation (4.8).

Second, we observe that the mappings $\vec{\chi}_i$ are only used between times τ_{i-1} and τ_i. Therefore, we can apply the CM method using only one mapping $\vec{\chi}$

in place of all the different mappings $\vec{\chi}_i$. To do so, solve equation (4.3a) for $\vec{\chi}$ instead of $\vec{\chi}_i$ and reset $\vec{\chi}(\vec{x}) = \vec{x}$ when a remapping time τ_i is reached.

The resulting procedure is described in algorithm 5

Algorithm 5 The characteristic mapping method

Define $\vec{\chi}(\vec{x}) = \vec{x}$ on a coarse N_c^d grid, $\vec{\chi}_0(\vec{x}) = \vec{x}$ on a fine N_f^d grid. S_0 is given.
for $t = 0$ to T **do**
  Advect $\vec{\chi}$ using the GALS method and advect particles using RK3 (eq. (4.3a) and (4.4a)).
  if $M_1(\vec{\chi}) > \mathcal{E}_1$ **then**
    $\vec{\chi}_0^{\text{temp}}(\vec{x}) \leftarrow \vec{\chi}_0(\vec{\chi}(\vec{x}))$
    $\vec{\chi}(\vec{x}) \leftarrow \vec{x}$
    $\hat{x}_p \leftarrow \hat{x}_p^0$
    if $M_2(\vec{\chi}_0^{\text{temp}}) > \mathcal{E}_2$ **then**
      $N_f^d \leftarrow (2N_f)^d$
      $\vec{\chi}_0(\vec{x}) \leftarrow \vec{\chi}_0(\chi(\vec{x}))$
    else
      $\vec{\chi}_0^{\text{coarse}}(\vec{x}) \leftarrow \vec{\chi}_0(\vec{\chi}(\vec{x}))$ on a grid 2 times coarser
      if $M_2(\vec{\chi}_0^{\text{coarse}}) < \mathcal{E}_2$ **then**
        $N_f^d \leftarrow (\frac{N_f}{2})^d$
        $\vec{\chi}_0(\vec{x}) \leftarrow \vec{\chi}_0(\vec{\chi}(\vec{x}))$
      else
        $\vec{\chi}_0(\vec{x}) \leftarrow \vec{\chi}_0^{\text{temp}}(x)$
      end if
    end if
  end if
end for

4.3 Numerical examples

In this section, we evaluate the accuracy and performance of the CM method. We use a grid of N_c cells per dimension for the advection of $\vec{\chi}$ and a grid of N_f cells per dimension to store $\vec{\chi}_0$. We will often compare the CM method to the GALS method, which will use a single grid having N_g cells per dimension.

We present standard benchmark tests in 2D and 3D in section 4.3.1 and 4.3.2. We then apply the CM method to more complicated sets in section 4.3.4 and 4.3.3. Finally, we present accuracy and efficiency results for the method in section 4.3.5.

4.3.1 2D swirl test

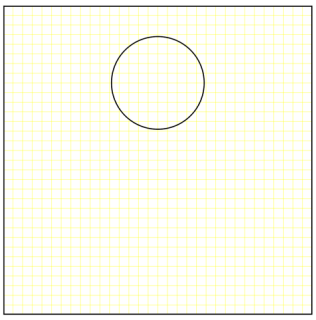
We apply the characteristic mapping method to the 2D case of the vector field $\vec{u}(\vec{x},t) = \{u(x,y,t), v(x,y,t)\}$ defined by

$$u\left(x,y,t\right) = \cos\left(\frac{\pi t}{A}\right)\sin^2\left(\pi x\right)\sin\left(2\pi y\right) \tag{4.12a}$$

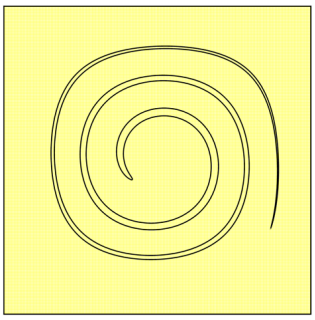
$$v\left(x,y,t\right) = -\cos\left(\frac{\pi t}{A}\right)\sin^2\left(\pi y\right)\sin\left(2\pi x\right) \tag{4.12b}$$

with $A = 16$ in the domain $[0,1]\times[0,1]$. This is the same vortex in a box as in section 3.2.1. The initial set is a circle of radius 0.15 centered at $(0.5, 0.75)$ represented by a level set function. From times $t = 0$ to $t = 16$, this flow will stretch the circle in a thin swirl and return it back to its original position. We use a $N_c^2 = 32^2$ grid for the advection and a dynamic-sized fine grid for the remapping. This fine grid starts at $N_f^2 = 32^2$ and is refined to a maximum resolution of $N_f^2 = 512^2$. The remapping and resizing tolerances were set to $\mathcal{E}_1 = 5\times10^{-6}$ and $\mathcal{E}_2 = 10^{-4}$. The results are shown in figure 4–3.

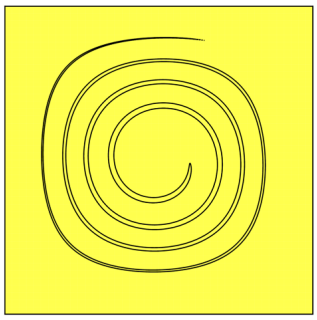
The initial identity transformation is perfectly represented on the 32×32 grid, but as time advances, the flow creates very fine structures. The transformation at $t = 8$ would definitely not be well represented on a 32×32 grid. But since the grid is adaptative, the remapping grid is refined as the transformation stretches the initial set. The grid for $\vec{\chi}_0$ reaches a size of 512×512 at $t = 8$, which is sufficient to represent correctly the difficult transformation induced by the vector field. Then as the spiral goes back to its original shape, the grid coarsens from $t = 8$ to $t = 16$. At this final time, the



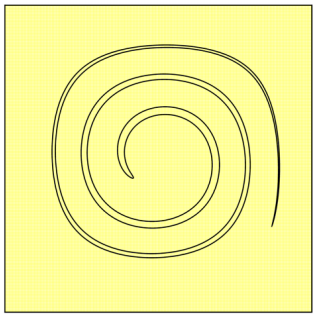
(a) $t = 0, N_f = 32$



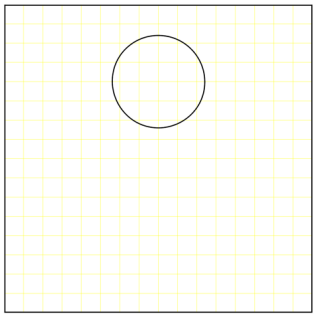
(b) $t = 4, N_f = 256$



(c) $t = 8, N_f = 512$



(d) $t = 12, N_f = 256$



(e) $t = 16, N_f = 16$

Figure 4–3: 2D swirl test using the characteristic mapping method with dynamic grid (i.e. N_f varies in time). Advection is done on a $N_c^2 = 32^2$ grid and remapping is done on a fine grid of maximal size $N_f^2 = 512^2$.

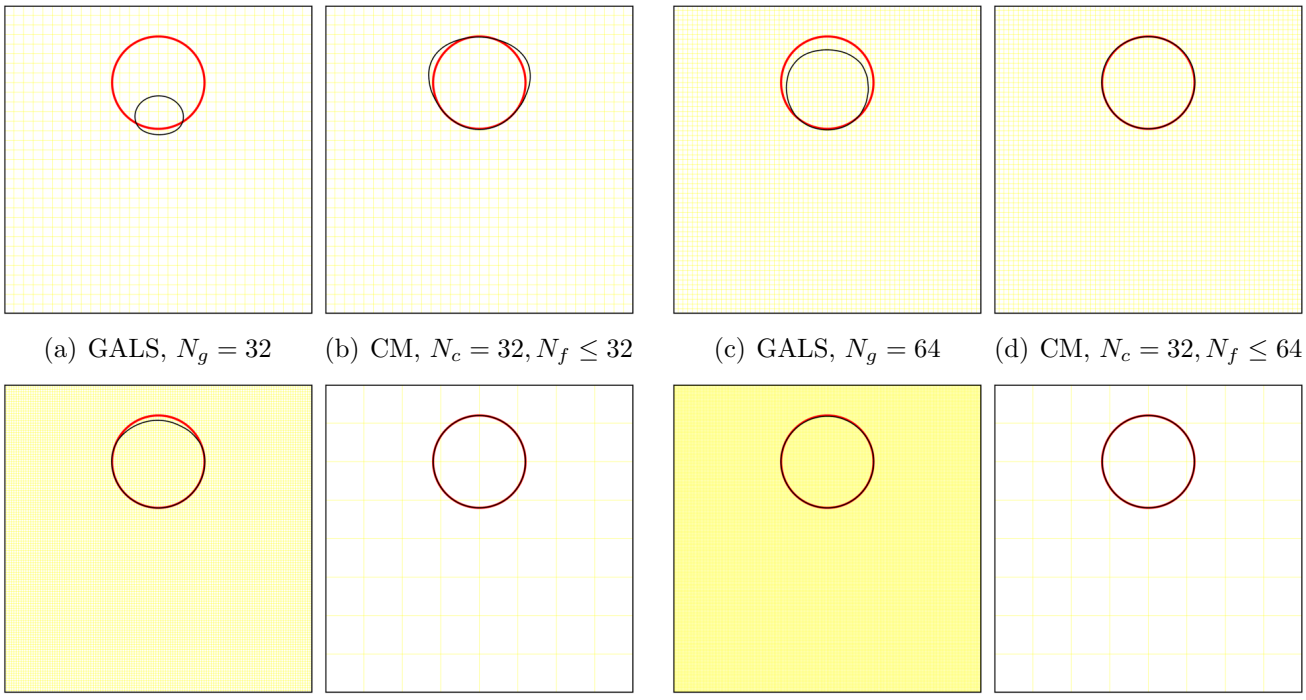
Computational Time (sec.)				
Method	N_f or N_g			
	32	64	128	256
GALS	2	12	92	727
CM with $N_c = 32$	6	7	9	11

Table 4–1: Time comparison (in seconds) of the GALS and CM methods for the swirl test (same test as in figure 4–4). The GALS method is advected on the given grid sizes (N_g). The CM method uses a $N_c^2 = 32^2$ grid for advection in all four cases, but the remapping is done on grids of the given sizes (N_f).

transformation only needs a 16×16 grid to be correctly represented. We see that the initial circle is recovered at the final time, even though the advection was computed on the rather coarse 32×32 grid and only a few remapping steps involved finer grid calculations.

We use a similar test to compare the characteristic mapping method with the standard GALS method. We use the same initial set and transport it in the vector field (4.12a)-(4.12b) with $A = 8$ until $t = 16$, which corresponds to the set being stretched and returned to its original position twice. We make four different tests by modifying the grid sizes. For the CM method, we use a $N_c^2 = 32^2$ for *all* tests but we set the maximal grid size to different values. We start for each of the four tests with $N_f = 32$ and allow the grid to be refined up to $N_f = \{32, 64, 128, 256\}$ respectively. For the GALS method, we fix the grid resolution to be the same as for the finest possible remapping grid of the CM method, i.e. $N_g = \{32, 64, 128, 256\}$. Results are shown in figure 4–4.

We see that for a given grid size, the characteristic mapping method gives a better solution than the GALS method. This is in part due to the fact that the CM method is initially advecting a linear transformation ($\vec{\chi}(t = 0, \vec{x}) = \vec{x}$) while the GALS method is advecting the more complicated level set function representing the circle. This explains why the CM solution is more accurate



(a) GALS, $N_g = 32$ (b) CM, $N_c = 32, N_f \leq 32$ (c) GALS, $N_g = 64$ (d) CM, $N_c = 32, N_f \leq 64$

(e) GALS, $N_g = 128$ (f) CM, $N_c = 32, N_f \leq 128$ (g) GALS, $N_g = 256$ (h) CM, $N_c = 32, N_f \leq 256$

Figure 4–4: Comparison of the GALS and CM methods for different grid sizes. For the CM method, advection is always done on a $N_c^2 = 32^2$ grid but the remapping grid is dynamically refined up to $N_f = \{32, 64, 128, 256\}$ in figures (b), (d), (f) and (h) respectively. The GALS uses the fixed grid $N_g = \{32, 64, 128, 256\}$ in figures (a), (c), (e) and (g) respectively. The computed solution is shown in black and the exact solution is shown in red.

even if the remapping is done on the same grid as the advection grid (as in figures 4–4(b) compared to figure 4–4(a)).

We also see that when the remapping grid is fine enough (as in figures 4–4(f) and 4–4(h)) the CM method is able to represent the transformation rather accurately and no significant error is accumulated. Therefore, the final transformation stored in $\vec{\chi}_0$ is smooth and very close to the identity transformation, which is represented sufficiently well on an 8×8 grid with our choice of $\mathcal{E}_1$ and $\mathcal{E}_2$. If the remapping grid is limited in its refinements as in 4–4b) and 4–4d), more error accumulates over time and the final transformation is not as close to the identity transformation, and thus a finer grid has to be maintained to represent the computed transformation.

A particularly attractive feature of the CM method is how computational time is in a sense optimized by the separation between coarse grid advection calculations (frequent but cheap) and fine grid storage operations on $\vec{\chi}_0$ (costly but infrequent). Table 4–1 compares the computational times for the GALS and CM method. For very coarse grids, the CM method is slower because it has to solve an advection problem for each dimension of the transformation. But for the more interesting case of an interpolation grid that is significantly finer than the advection grid (e.g. for a 256×256 remapping grid and a 32×32 advection grid), the CM method is clearly faster. The efficiency aspect is studied in more details in section 4.3.5.

4.3.2 3D deformation field

Since the characteristc mapping method solves (4.3a) independently for each dimension, there is no difficulty in implementing the method in any number of spatial dimension. We apply the CM method to the 3D case of a sphere of radius 0.15 centered at $(0.35, 0.35, 0.35)$ in the domain $[0, 1]^3$ and deformed

under the vector field $\vec{u}(\vec{x},t) = \{u(x,y,z,t), v(x,y,z,t), w(x,y,z,t)\}$, where

$$u(x,y,z,t) = 2\cos\left(\frac{\pi t}{2}\right)\sin(\pi x)^2\sin(2\pi y)\sin(2\pi z) \tag{4.13a}$$

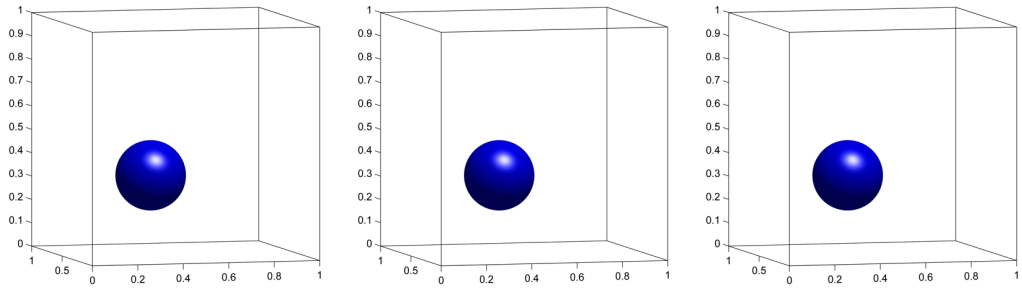
$$v(x,y,z,t) = -\cos\left(\frac{\pi t}{2}\right)\sin(2\pi x)\sin(\pi y)^2\sin(2\pi z) \tag{4.13b}$$

$$w(x,y,z,t) = -\cos\left(\frac{\pi t}{2}\right)\sin(2\pi x)\sin(2\pi y)\sin(\pi z)^2. \tag{4.13c}$$

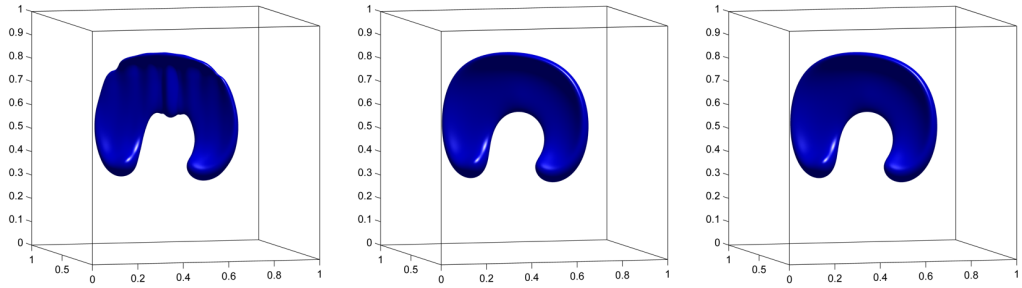
This standard velocity field is the same as in section 3.2.1. The advection of $\vec{\chi}$ is done on a $N_c^3 = 16^3$ grid, and the remapping of $\vec{\chi}_0$ is done on a fixed $N_f^3 = 128^3$ grid with a remapping tolerance $\mathcal{E}_1 = 10^{-4}$. We compare the results with and without remapping to demonstrate the benefits of the remapping step. We also compare the CM method to the GALS method computed on a $N_g^3 = 128^3$ grid. Results for $t = 0$, $t = 1$ and $t = 2$ for the three cases are shown in figure 4–5.

The surface at $t = 2$ is expected to be identical to the surface at $t = 0$ (sphere) due to the $\cos(\frac{\pi t}{2})$ term in equations (4.13). This vector field causes the sphere to be stretched along the $y = 1 - x$ plane and thus creates very fine structures that cannot be represented on the coarse 16^3 grid. We see from figure 4–5(d) that without remapping, those structures are indeed not well represented and oscillations are observed on the scale of the coarse grid. Those oscillations distort the surface for all subsequent times, and the final shape in figure 4–5(g) is significantly different than the initial sphere.

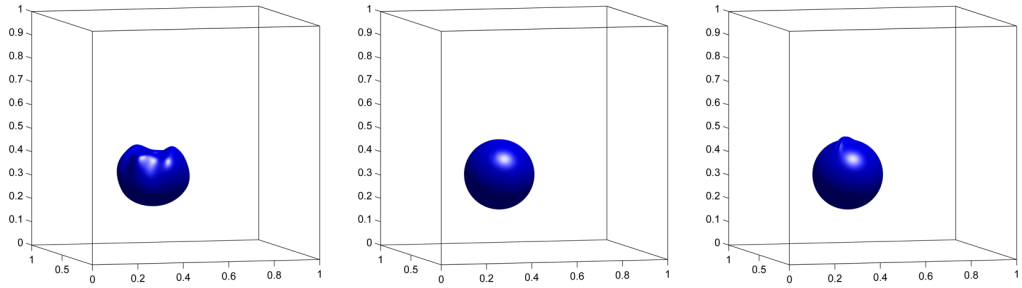
When using the remapping on the fine grid, the fine structures caused by the deformation field can be accurately represented as a result of storing $\vec{\chi}_0$ on a $N_f^3 = 128^3$ grid. In figure 4–5(e), the width of the stretched surface is of the order of the fine grid's cell width, therefore causing no major representation issues. At $t = 2$ (figure 4–5(h)), the surface is visually identical to the initial sphere.



(a) CM without remapping, t=0 (b) CM with remapping, t=0 (c) GALS, t=0



(d) CM without remapping, t=1 (e) CM with remapping, t=1 (f) GALS, t=1



(g) CM without remapping, t=2 (h) CM with remapping, t=2 (i) GALS, t=2

Figure 4–5: 3D deformation of a sphere. Left: CM on a $N_c^3 = 16^3$ grid without remapping. Middle: CM on a $N_c^3 = 16^3$ grid with remapping on a $N_f^3 = 128^3$ grid. Right: GALS on a $N_g^3 = 128^3$ grid.

Method	Computational Time (sec.)
CM without remapping	267
CM with remapping	1430
GALS	29687

Table 4–2: Time comparison (in seconds) of the GALS and CM methods with and without remapping for the 3D deformation test of figure 4–5. We use $N_c = 16$ and $N_f = N_g = 128$.

The GALS method also uses a $N_g^3 = 128^3$ grid and is therefore able to capture the same level of detail as the characteristic mapping method with remapping. The main difference between both methods is that the error due to interpolation present in the CM method is kept in check due to the remapping strategy. This can be observed by comparing figures 4–5(e) and 4–5(f) where the solution is not significantly different, while after a longer time (figures 4–5(h) and 4–5(i)), the accumulated interpolation error shows as a small kink for the GALS case.

The CM method is also interesting for its computational efficiency. Table 4–2 compares the time taken to compute the three tests of figure 4–5. As expected, the time taken by the CM method without remapping is small, but the results are not accurate. The time taken by the CM method with remapping is approximately 5 times larger than without remapping, but the results we obtain are almost perfect. The GALS method takes about 20 times longer than the CM method with remapping to compute its solution, and the results are worse. This again shows the superior efficiency of the CM method over the GALS method. This aspect is studied in more details in section 4.3.5.

4.3.3 Complicated sets

We show in this section that the characteristic mapping method can be used to advect arbitrarly complex sets by presenting two different tests. For

the first test, we take the deformation field $\vec{u}(\vec{x},t) = \{u(x,y,t), v(x,y,t)\}$ given by

$$u\left(x,y,t\right) = \cos\left(\frac{\pi t}{A}\right)\left(-\frac{1}{4}L\left(x\right)\sin^2\left(\frac{3}{2}\pi x\right)\sin\left(4\pi y\right) + \frac{3}{4}R\left(x\right)\left(x-0.5\right)\right) \tag{4.14a}$$

$$v\left(x,y,t\right) = \cos\left(\frac{\pi t}{A}\right)\left(\frac{1}{4}L\left(x\right)\sin^2\left(2\pi y\right)\sin\left(3\pi x\right) + \frac{3}{4}R\left(x\right)\left(y-0.5\right)\right) \tag{4.14b}$$

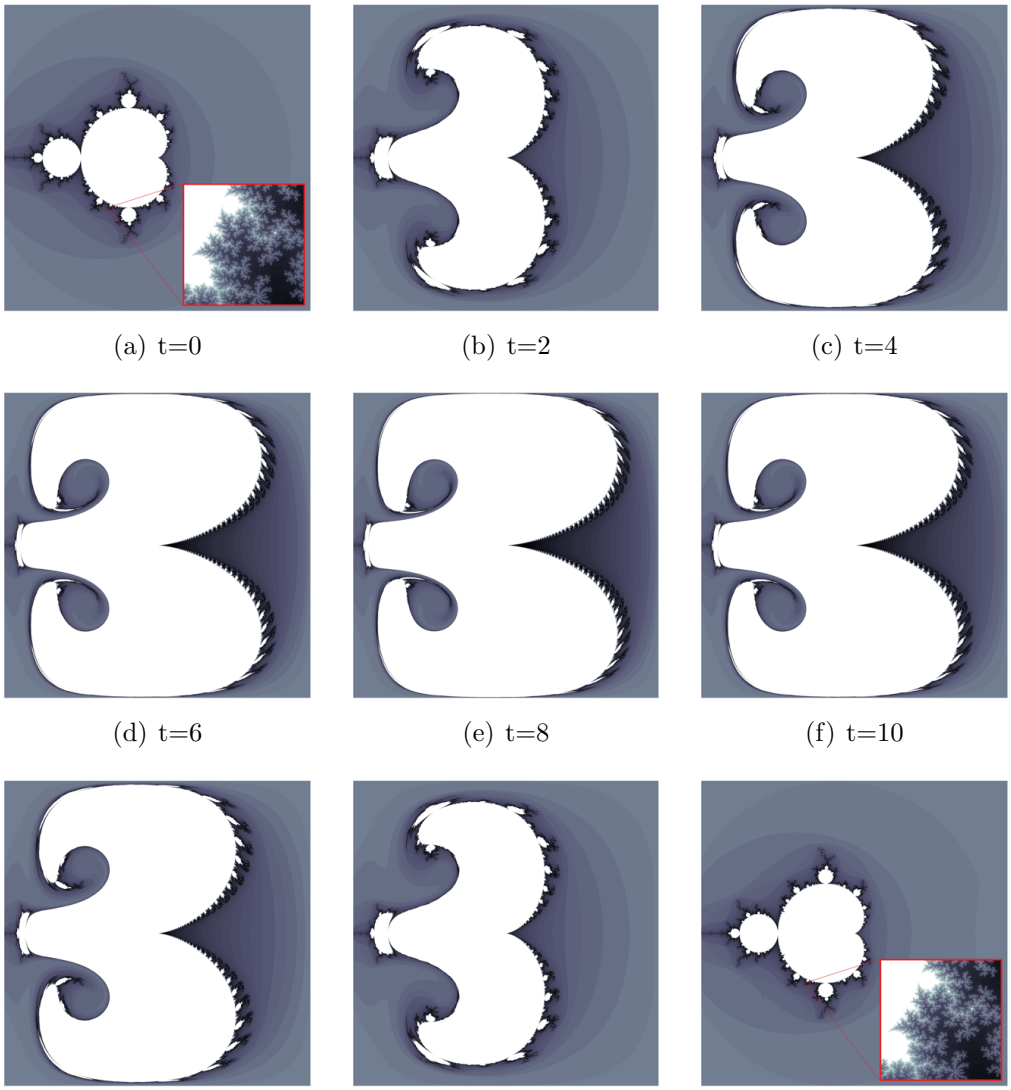
with A=16 and where L and R are smooth weight functions defined by

$$R\left(x\right) = \sin\left(\pi\left(4x^2 - 5x^3 + 2x^4\right)\right)\sin\left(\pi y\right) \tag{4.14c}$$

$$L\left(x\right) = \sin\left(\pi\left(1-x\right)^3\right)\sin\left(\pi y\right)\sin\left(\frac{3}{2}\pi x\right)\sin^2\left(2\pi y\right). \tag{4.14d}$$

This vector field causes the left region to swirl into two vortices and the right region to expand around the center of the domain. We apply this vector field the the Mandelbrot set and compute the advection of the set with the CM method. We use $N_c = 32$ for the coarse grid, $N_f = 1024$ for the fine grid and a remapping tolerance $\mathcal{E}_1 = 10^{-7}$. Note that we did not use a dynamic grid resolution for this test. The results at times $t = 0$ to $t = 16$ are shown in figure 4–6.

As the set evolves, the centeral region is enlarged to the right and additional details of the set appear. Since we know the initial set with arbitrary precision, there is no problem capturing all those details. Doing a similar test with usual level set methods would be impossible since the level set function describing the Mandelbrot set is extremely hard to represent numerically, while representing the smooth transformation with level set functions is very easy. Also, having a numerical representation of the Mandelbrot set on a fine grid



(a) t=0 (b) t=2 (c) t=4

(d) t=6 (e) t=8 (f) t=10

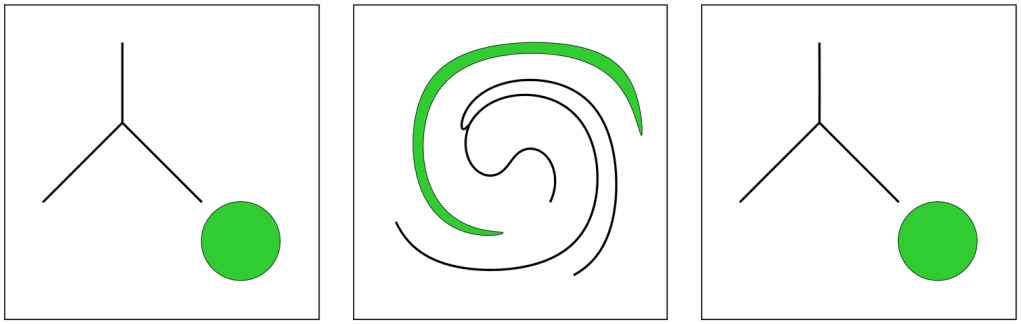
(g) t=12 (h) t=14 (i) t=16

Figure 4–6: 2D deformation of the Mandelbrot set computed with the CM method with an advection grid of $N_c^2 = 32^2$ and a fixed remapping grid of $N_f^2 = 1024^2$. The zoomed regions in figures (a) and (i) correspond to a single grid cell of the fine grid.

would not allow to represent the new details that appear during the transformation. Apart from a few exceptions where the transformation has developed structures that fall under the grid size during the transformation, we recover the initial surface with great precision.

For the first and last frame, we enlarged one of the grid cells of the 1024×1024 grid by evaluating the Hermite interpolant at many additional locations. This cell is in the region that is most affected by the deformation in the left side of the domain. We see that even though this is a difficult test that causes a lot of stretching, the identity transformation is recovered accurately. Even by using the computed transformation to draw some very fine details of the set (i.e. much finer than the fine grid size), no significant qualitative difference is oberved.

The second test involves open curves. Numerically, these objects are challenging since they are hard to precisely represent on a grid. Still, we would like to have a formulation that allows computations of normals and curvature on a regular grid. A simple solution is to represent such open curves using two set functions, where one is used as a mask function. Figure 4–7 shows an example where we advect three independent open curves forming a triple point with open ends, along with a closed circle. These objects are transported in the swirling velocity field (4.12) with $A = 4$. Each part of the curve is defined as a straight line enclosed in a mask region. Both the line and the mask are defined by simple level set functions. Since the CM method decouples the advection from the initial set, we can advect those seven set functions (three planes, three masks and one cone) at the same time at the only additional cost of a single interpolation evaluation step on the fine grid for every set function, only when plotting is required.



(a) t=0 (b) t=2 (c) t=4

Figure 4–7: 2D deformation of open curves. Each of the three branches (black) is defined by a linear level-set function enclosed in a mask set, also defined by another level-set function. The circle is also represented by a level set and is colored green to keep track of its interior.

4.3.4 Triple and quadruple points mosaic

We show another test to emphasize the advantages of using a diffeomorphism formulation. We take the $[0,1] \times [0,1]$ domain with periodic boundary and subdivide it into multiple regions. This kind of initial situation can arise, for instance, in simulation of multiphase flows. A difficulty of these simulations resides in the triple point caused by the junction of three different fluids. Triple points are hard to represent using a single level set function, but can be represented as multiple piecewise level sets. In the context of the CM method, this is not a problem since we can transport many functions at the same time.

The velocity field used for this test is

$$u\left(x,y,t\right) = \cos\left(\frac{\pi t}{2}\right)\cos\left(2y\pi + 2\sin\left(2\cos^2\left(\frac{\pi t}{2}\right)\right)\right) \tag{4.15a}$$

$$v\left(x,y,t\right) = \cos\left(\frac{\pi t}{2}\right)\sin\left(2x\pi + 2\sin\left(\cos^2\left(\frac{\pi t}{2}\right)\right)\right) \tag{4.15b}$$

and we used $N_c = 32$ and $N_f = 512$. It took 65 seconds to compute the 2048 steps of this simulation on a single 3.0GHz CPU, taking 0.008 seconds for a regular step and 0.45 seconds when remapping, which was necessary about every 12 steps on average. We also highlight three regions in the domain. These

regions represent two triple points and one quadruple points. The dashed circles are transported in the flow as passive particles. Doing so emphasizes the fact that the intersections follow the right paths. Also, they show that even if some regions are transformed into very thin filaments, we can still track them in the flow. For instance, the highlighted region around the purple-yellow-green intersection (bottommost circle in figures 4–8(a) and (i), topmost in other subfigures) shows that even if we visually lose the intersection at time $t = 1$, it returns to its original position at time $t = 2$.

4.3.5 Computational efficiency

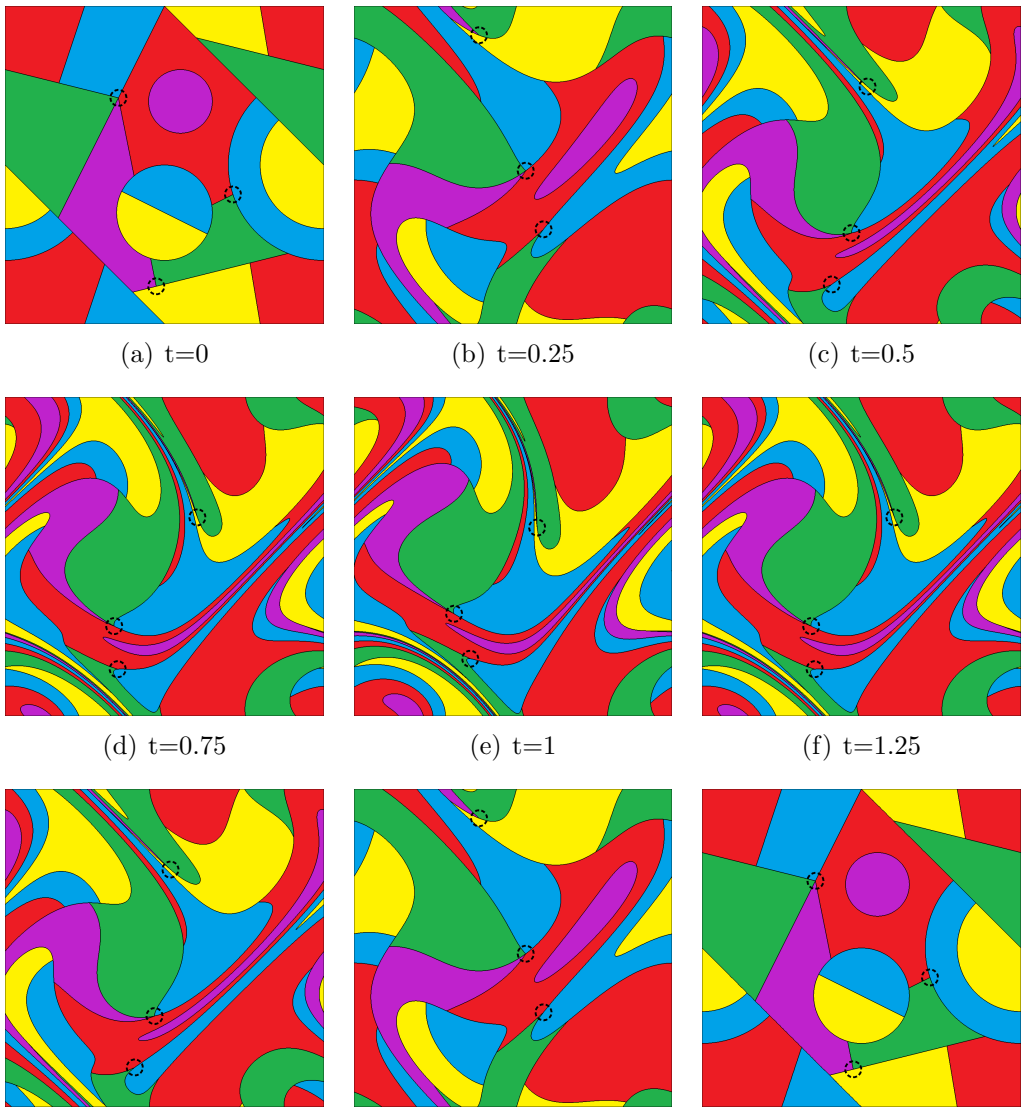
We analyse here the computational effort required by the CM method and compare it to the cost of the GALS method. The figures presented in this section use the swirl test presented in section 3.2.1 wth $A = 8$ and a final time $T = 16$. To facilitate the interpretation of the results, we do not use a dynamic fine grid and set $\triangle t = 1/N_f$. We use $N_c = 32$ for all computations, and the values of N_f and N_g vary on the graphs. We compare those costs for one advection step.

For the GALS method, the cost is that of tracing back the footpoints, and then evaluating the interpolant at those locations. This cost can be expressed as

$$\text{cost GALS} = \underbrace{C_1 N_g^d}_{\text{footpoints}} + \underbrace{C_2 N_g^d}_{\text{interpolation}} \tag{4.16}$$

for some constants C_1 and C_2, and where d denotes the dimension and N_g is the number of grid cells for each spatial dimensions, as before.

For the CM method, we also need to trace back the footpoints and interpolate the function, but that is done on the coarse N_c^d grid and has to be



(a) t=0 (b) t=0.25 (c) t=0.5

(d) t=0.75 (e) t=1 (f) t=1.25

(g) t=1.5 (h) t=1.75 (i) t=2

Figure 4–8: 2D deformation of a periodic mosaic pattern. The advection grid has $N_c^2 = 32^2$ with a fixed remapping grid of $N_f^2 = 1024^2$. Three regions are highlithed and tracked as passive particles.

done independently for each dimension. Additionally, we need to advect the particles and sometimes do a remapping step. If we do a remapping after M advection steps, the cost for the CM method can be expressed as

$$\text{cost CM} = \underbrace{dC_1N_c^d}_{\text{footpoints}} + \underbrace{dC_2N_c^d}_{\text{interpolation}} + \underbrace{C_1\gamma N_c^d}_{\text{particles}} + \underbrace{(1/M)dC_2(N_f^d + N_c^d)}_{\text{remapping}} \tag{4.17}$$

where γ is the number of particles per cell. Note that the remapping part involves two terms because of the composition in equation (4.8). Note also that M depends on $\vec{u}$ and $\mathcal{E}_1$ in a non-trivial way.

To compare the efficiency of both methods, we take $N_g = N_f$. We note directly from (4.16) and (4.17) that if N_c is small enough compared to N_f and M is large enough, the CM method will perform faster than the GALS method. We also note that the remapping term is crucial in the analysis of computational time since it is the only one that contains N_f. This makes it a costly part of the method since we are interested in regimes where N_f is much larger than N_c. This implies that M plays an important role in the efficiency of our method. M depends monotonically on $\mathcal{E}_1$ because taking a larger $\mathcal{E}_1$ increases the value of M (remapping steps are less frequent), but doing so also causes the error of the computed transformation to increase. Therefore, for $\vec{u}, N_c$ and N_f given, the choice of $\mathcal{E}_1$ determines the trade-off between accuracy and efficiency. This also suggests that for a given value of the error, there is an optimal value for $\mathcal{E}_1$.

Figure 4–9 shows computational times against grid widths ($1/N_f$) for different values of $\mathcal{E}_1$. The figure confirms that reducing $\mathcal{E}_1$ increases the computational time. We also see that if the N_f grid is too coarse, the GALS method performs faster. This is to be expected since the CM method has to compute an advection step for each dimension. Therefore, if the N_c grid is of

similar size as the N_f grid, the CM method requires more computations, as is seen in equation (4.16). But if N_f is significantly larger than N_c, the multiple advection steps are much cheaper to compute than the single GALS step.

Figure 4–10 shows the L_2 error of the solution against grid widths ($1/N_f$) for different values of $\mathcal{E}_1$. The first thing to notice is that the curves for $\mathcal{E}_1 = 10^{-4}$ and $\mathcal{E}_1 = 10^{-5}$ change their behavior depending on the grid size. Since $\mathcal{E}_1$ represents the interpolation error made before each remapping step, we expect the error curves to stagnate when the error reaches the corresponding values of $\mathcal{E}_1$. Since $\triangle t$ scales with the grid size, the error due to advection continues to decrease, which explains a slower decrease of the error in the graphs beyond this critical point. Secondly, we see that for regions where curves have not yet reached their critical error, a smaller value of $\mathcal{E}_1$ gives a bigger error. This is expected, because a bigger value for $\mathcal{E}_1$ implies more remapping steps, and these remapping step each produce an interpolation error.

More importantly, figure 4–11 shows the computational times against the L_2 error for different values of $\mathcal{E}_1$. From this figure we clearly see the influence of $\mathcal{E}_1$ on computational time and observe that there is an optimal $\mathcal{E}_1$ for a given global error of the solution. If we denote the global error in the solution by E, we observe once again that taking $\mathcal{E}_1 > E$ is not an efficient choice. Also, for values of $\mathcal{E}_1$ greater than E, smaller values of $\mathcal{E}_1$ give better computational times. Therefore, our results for this test suggest that a criterion for choosing $\mathcal{E}_1$ optimally is to take $\mathcal{E}_1$ to be the desired global error of the final solution.

Another major advantage of the CM method is that it is easily parallelizable. The advection of each dimension of the transformation is independent, and multiple transported interfaces can be computed separately. Moreover, since the advection of each grid point is done using the local GALS scheme,

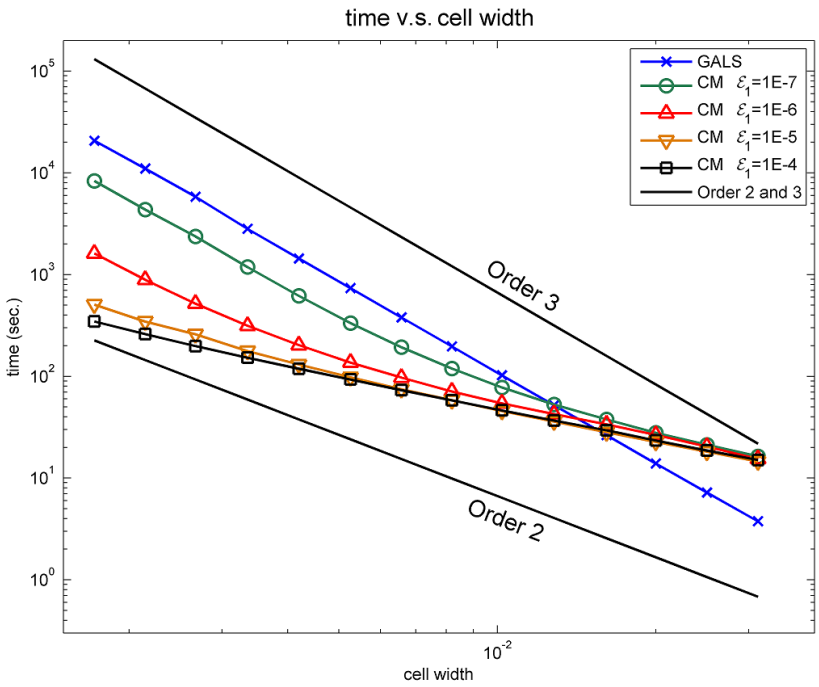


Figure 4–9: Time v.s. cell width for the CM method.

the value of the transformation at each grid point can be computed independently as well. Also, the remapping step is easily parallelizable beacause it only requires a Hermite interpolation, which is a local operation. It is not the aim of this paper to investigate the parallel implementation of the CM method, but it should be noted that future work in this direction is promising.

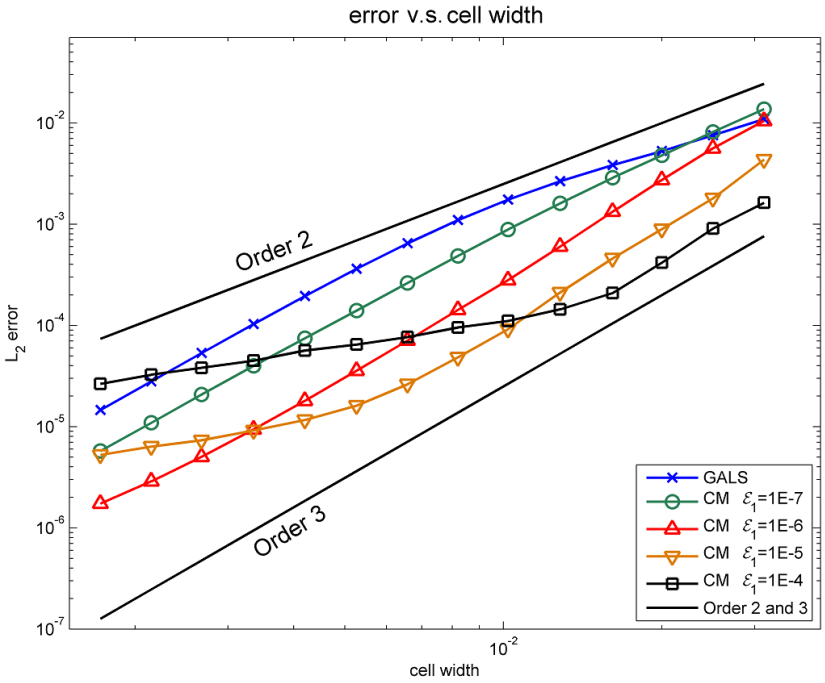


Figure 4–10: Error v.s. cell width for the CM method.

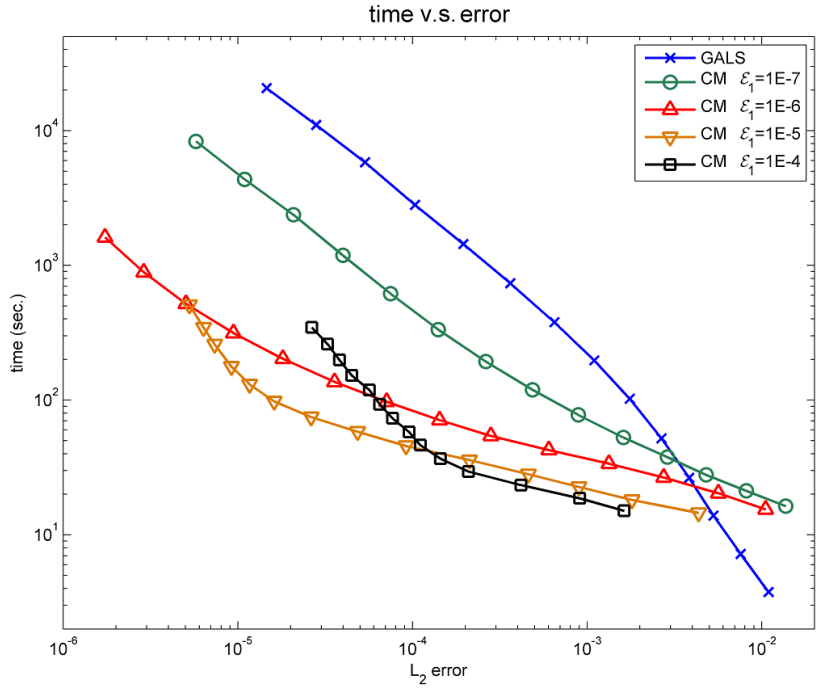


Figure 4–11: Time v.s. error for the CM method.

CHAPTER 5
Application to specific partial differential equations

5.1 Navier-Stokes equations

Fluid simulations are a very important application of set transport. Fluids are governed by the *Navier-Stokes* equations. For an incompressible fluid living in a domain U, the equations are

$$\partial_t \vec{u} + \vec{u} \cdot \nabla \vec{u} = -\frac{\nabla P}{\rho} + \frac{\mu}{\rho} \nabla^2 \vec{u} \tag{5.1}$$

$$\nabla \cdot \vec{u} = 0 \tag{5.2}$$

where

$\vec{u}$:=Velocity of the fluid.

P :=Pressure of the fluid.

ρ :=Density of the fluid.

μ :=Dynamic viscosity of the fluid.

Boundary conditions also have to be added to dictate the behavior of the fluid on the boundary ∂U of the domain U. We will use here a no slip condition, i.e. $\vec{u} = 0$ on ∂U. Note that since the fluid is incompressible, ρ is constant.

We are interested in simulating multiphase flows. A simple example is the simulation of a volume of water moving in air. We will deal with the 2D case for simplicity and use the notation $\vec{u}(x, y) = (u(x, y), v(x, y)) \in \mathbb{R}^2$. A standard situation is represented in figure 5–1. We will refer to this figure in what follows.

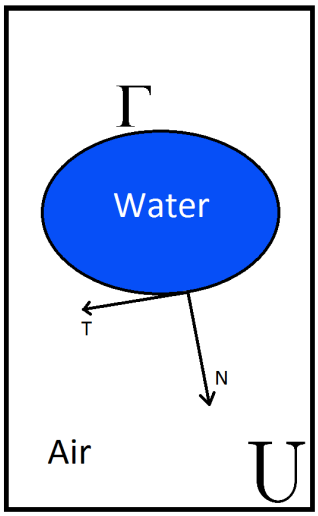


Figure 5–1: Multiphase fluid. A volume of water moves in air in a closed domain U.

Our goal is to track the boundary Γ between air and water. Evolving this boundary in time is enough to describe how both the air and the water move. Many additional conditions have to be imposed on each point of this boundary for it to move in a physically correct way. We list them here with a brief explanation. We use the jump notation $[Q] := Q_{\text{air}} - Q_{\text{water}}$ for any quantity Q.

- $[P - 2\mu\,(\nabla u \cdot N, \nabla v \cdot N) \cdot N] = \sigma\kappa$

  where σ is the surface tension coefficient and κ is the curvature of the boundary at a given point. This adds surface tension effects to the flow.
- $[\mu(\nabla u \cdot N, \nabla v \cdot N) \cdot T + \mu(\nabla u \cdot T, \nabla v \cdot T) \cdot N] = 0.$

  This indicates that the stress on the surface must be zero.
- $[u] = [v] = 0.$

  The flow must be continuous across the surface.
- $[\nabla u \cdot T] = [\nabla v \cdot T] = 0.$

  No slip condition for a moving boundary.

- $[\partial_t u + \vec{u} \cdot \nabla u] = [\partial_t v + \vec{u} \cdot \nabla v] = 0.$

  The flow is transported with the boundary.

To solve the Navier-Stokes equations, we split in time equations (5.1) and (5.2) by using a *projection method*. This breaks the equation into smaller parts that are easier to solve. The method we present here is that of Kang et al. [14]. We will use the notation $\vec{u}^k$ to denote the velocity at time step k, and similarly for all other variables. Start with a known velocity field, possibly $\vec{u}^0 = 0$. First, take the pressure term out in equation (5.1) and discretize it using a temporary variable $\vec{u}^*$ in place of $\vec{u}^{k+1}$.

$$\frac{\vec{u}^* - \vec{u}^k}{\Delta t} + \vec{u}^k \cdot \nabla \vec{u}^k = \frac{\mu}{\rho} \nabla^2 \vec{u}^k \tag{5.3}$$

$$\Rightarrow \vec{u}^* = \vec{u}^k + \Delta t \left(\frac{\mu}{\rho} \nabla^2 \vec{u}^k - \vec{u}^k \cdot \nabla \vec{u}^k \right). \tag{5.4}$$

Note that $\vec{u}^*$ can be computed easily since $\vec{u}^k$ is known. The true value of $\vec{u}^{k+1}$ will then be computed by adding the pressure term.

$$\frac{\vec{u}^{k+1} - \vec{u}^*}{\Delta t} = -\frac{\nabla P^k}{\rho} \tag{5.5}$$

$$\Rightarrow \vec{u}^{k+1} = \vec{u}^* - \Delta t \frac{\nabla P^k}{\rho}. \tag{5.6}$$

But the pressure P^k is not known. To compute it, we take the divergence of equation (5.5) to get

$$\nabla \cdot \left(\frac{\vec{u}^{k+1} - \vec{u}^*}{\Delta t} \right) = -\nabla \left(\frac{\nabla P^k}{\rho} \right) \tag{5.7}$$

$$\Rightarrow \nabla^2 P^k = \frac{\rho}{\Delta t} \nabla \cdot \vec{u}^* \tag{5.8}$$

where we used $\nabla \cdot \vec{u}^{k+1} = 0$. This assumption projects the solution on a divergence free space. Equation (5.8) requires to solve a Poisson equation with jump conditions on the entire domain. In order to have a completely

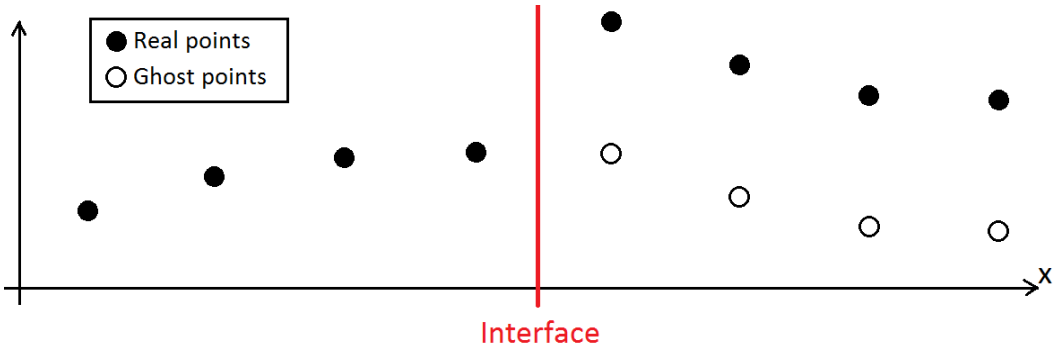


Figure 5–2: Ghost fluid method. A derivative is to be evaluated on a grid point just left of the boundary. Using the jump conditions at the interface, we translate the values to the right of the interval to create ghost points. We then use those points to evaluate the derivative using a finite difference scheme.

explicit scheme in time, a time step is computed by solving equations (5.4), (5.8) and (5.6) in this order.

Note that because of the jump conditions on the water-air interface Γ, some of the spatial derivatives cannot be evaluated using simple finite difference schemes. We instead use the *ghost fluid method* [9] to evaluate those derivatives. Figure 5–2 illustrates this method in 1D where a derivative needs to be evaluated less than a grid point to the left of the interface. The right data points cannot be used to compute the derivative since they represent another branch of the function. But since we know what the jump is between the left and right values, we can adjust the right values to create *ghost points* and use those points to compute the derivatives using finite difference schemes.

Once the vector field is computed at a given time step, we can use any of the methods described in chapters 2, 3 and 4 to transport the interface between the air and the water. However, our attempts at solving the Navier-Stokes equations are not yet stable. Discontinuities in the solution make the method difficult to control. A discontinuity can be created for instance when two water droplets merge. More research needs to be done in order to fully understand

the behavior of our new methods in conjunction with the projection method described here. Still, we were able to do fluid simulations by simplifying the problem, as described in the next section.

5.2 Euler equations

The 2D Navier Stokes equations are greatly simplified by neglecting the viscosity term. Doing so gives the Euler equation

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} + \frac{\nabla P}{\rho} = 0 \tag{5.9}$$

$$\nabla \cdot \vec{u} = 0 \tag{5.10}$$

To further simplify the problem, we will look here at a single phase flow in a periodic box. We can rewrite equation (5.9) in a more convenient form. Define the *vorticity* $\omega := \nabla \times \vec{u}$, where $\nabla\times$ is the curl operator. By taking the curl of (5.9) and using various vector calculus identities, the Euler equation becomes

$$\nabla \times \frac{\partial \vec{u}}{\partial t} + \nabla \times (\vec{u} \cdot \nabla)\vec{u} + \underbrace{\frac{\nabla \times \nabla P}{\rho}}_{\text{curl of gradient is 0}} = 0 \tag{5.11a}$$

$$\Rightarrow \quad \frac{\partial(\nabla \times \vec{u})}{\partial t} + \nabla \times \left(\frac{1}{2}\nabla(\vec{u} \cdot \vec{u}) - \vec{u} \times (\nabla \times \vec{u}) \right) = 0 \tag{5.11b}$$

$$\Rightarrow \quad \frac{\partial \omega}{\partial t} + \underbrace{\nabla \times \nabla \left(\frac{\vec{u} \cdot \vec{u}}{2} \right)}_{\text{curl of gradient is 0}} - \nabla \times (\vec{u} \times \omega) = 0 \tag{5.11c}$$

$$\Rightarrow \quad \frac{\partial \omega}{\partial t} + \vec{u} \cdot \nabla\omega - \underbrace{\omega \cdot \nabla \vec{u}}_{\nabla\vec{u}\perp\omega} + \omega \cdot (\underbrace{\nabla \cdot \vec{u}}_{\nabla\cdot\vec{u}=0}) + \vec{u} \cdot (\underbrace{\nabla \cdot \omega}_{\text{div. of curl is 0}}) = 0 \tag{5.11d}$$

which gives the *vorticity equation*

$$\frac{\partial \omega}{\partial t} + \vec{u} \cdot \nabla\omega = 0. \tag{5.12}$$

This equation can be interpreted as a transport equation for the vorticity, meaning that the vorticity is simply transported in the flow. However, it

is very different from the standard transport equation because ω and $\vec{u}$ are coupled, which makes the problem non-linear. Still, an approximated solution of this equation can be obtained by decoupling ω and $\vec{u}$. Note also that the vorticity equation implicitely takes into account the condition (5.10), so it is not required to deal with the pressure term to impose a divergence-free flow as is otherwise done for example in [26].

To solve the vorticity equation, we use the CM method presented in chapter 4. The main reason for this is that the solution of (5.12) is known to be a diffeomorphism of the initial vorticity distribution [2]. Another reason is because of the decoupling of scales in the CM method. It allows the flow to be governed by the large vortices of the flow, while still representing the fine details of the simulation.

If we know $\vec{u}^k$, we can therefore compute ω^{k+1} using the CM method. But computing $\vec{u}^k$ from ω^k requires some explanations.

Since the flow has to satisfy $\nabla \cdot \vec{u} = 0$, it can be represented using a *stream function* ϕ such that

$$\vec{u} = \left(-\frac{\partial \phi}{\partial y}, \frac{\partial \phi}{\partial x} \right). \tag{5.13}$$

The fact that $\vec{u}$ is representable by such a stream function is justified by the Hodge decomposition. Conversely, representing $\vec{u}$ by a stream function ensures incompressibility by construction. This definition leads to the relation

$$\phi = -\Delta^{-1}\omega \tag{5.14}$$

where Δ^{-1} is the inverse of the Laplace operator. Equation (5.14) is easy to solve in Fourier space where it becomes

$$\hat{\phi} = \frac{\hat{\omega}}{k_x^2 + k_y^2} \tag{5.15}$$

where $\hat{\cdot}$ represents the Fourier transform, and k_x and k_y are the x and y Fourier coefficients. See [25] for details. To recover ϕ, an inverse Fourier transform has to be computed. We used the *FFTW* library [10] for all Fourier transforms.

With the separation of scales, ω is stored on a fine grid, but the velocity field is stored on the coarse grid. Therefore, the computations to recover $\vec{u}$ from ω are only required on the coarse grid, which significantly accelerates the algorithm. To do so, compute the Fourier transform of ω, and truncate $\hat{\omega}$ to keep only the low frequencies on the coarse grid. This process saves the information coming from the large vortices, and gets rid of the information of smaller details of ω when computing $\vec{u}$. Algorithm 6 summarizes the whole process.

Algorithm 6 Solving the vorticity equation

· The initial ω_0 is given on the fine grid.
for $t = 0$ to ∞ **do**
· Compute $\hat{\omega}$ on the fine grid.
· Truncate $\hat{\omega}$ to the coarse grid, keeping low frequencies.
· Compute $\hat{\phi}$ on the coarse grid using equation (5.15).
· Compute ϕ on the coarse grid using an inverse Fourier transform.
· Compute $\vec{u}$ on the coarse grid using (5.13).
· Advect ω with one step of the diffeomorphism approach of chapter 4 and remap if needed.
· $t \leftarrow t + \Delta t$.
end for

5.2.1 Results

Two cases are presented to analyse the method. Implementation of the algorithm was done in C++. The frames were created using the *FreeImage*[1] library.

5.2.2 Test case : three vortices

The initial state of vorticity ω_0 used for this test consists of two negative votices (i.e. clockwise curl) of strength 1 and one positive vortex (i.e. counter-clockwise curl) of strength 2. This initial condition is obtained by using a sum of gaussian functions as the initial vorticity. This initial condition is represented in the top left subfigure of figure 5–3, where the blue to red scale represents vorticity from -1 to 2. Figure 5–3 presents three sets of frames showing the evolution of the solution. From left to right and top to bottom, they are frames 0, 15, 30, 800, 815, 830, 2400, 2415 and 2430. The vorticity is stored on a fine grid with $N_f = 256$ cells per dimensions, and advection computations are done on a coarse grid having $N_c = 32$.

The behavior obtained is qualitatively as expected. The flow is governed by the large vortices, while the finer details are passively advected in the flow. Still, the frames show that a very high level of detail is maintained throughout the simulation. The combination of a fine grid and a bicubic interpolant allows to represent the fine details observed, for instance, in the subfigures of the bottom row of figure 5–3.

[1] http://freeimage.sourceforge.net

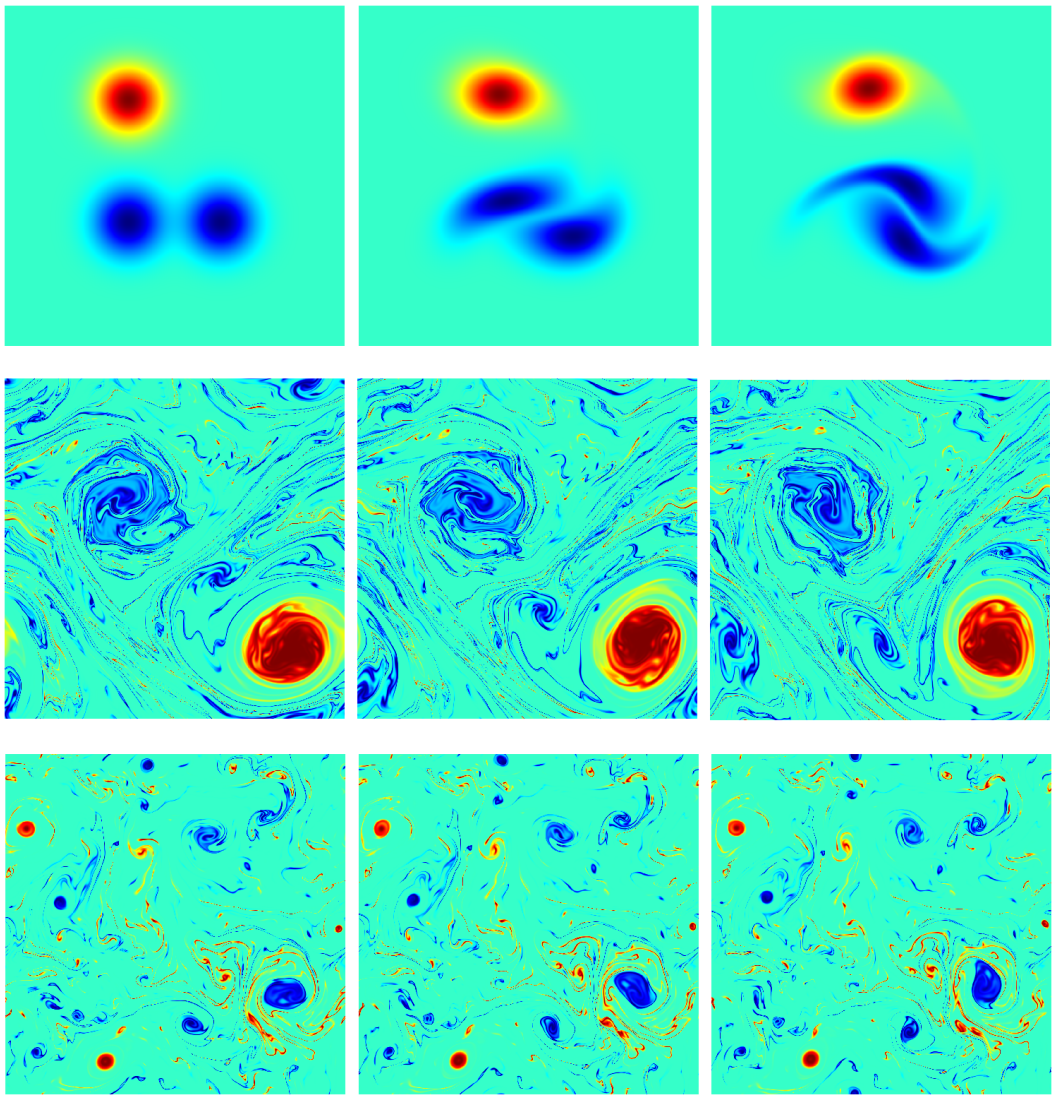


Figure 5–3: Evolution of three vortices at three sets of frames, evolving from left to right. The scale from blue to red represents vorticity from -1 to 2.

5.2.3 Fine structures

Even if the bicubic interpolant allows to have a high resolution representation of the solution, it has one major drawback. Since the bicubic Hermite interpolant can overshoot the data that is on the grid, it can lead to instabilities. The right column of figure 5–4 shows three frames of a simulation done using the bicubic interpolant, along with a zoom of the central region. The simulation starts with two negative vortices, which creates heavy compressions around the central region. The first figure shows a clean solution around the compression line. But as time advances, the second and third pictures show noises and instabilities around this line, caused by the bicubic interpolant overshooting the data. Since the cubic represents the diffeomorphism, instabilities will cause points to be mapped to far away places in the domain. This explains the seemingly random noise observed in the second and third figures of the right column of figure 5–4. Figure 5–5 shows the grid transformed by the cubic diffeomorphism for the same frames, and we see that the regions where the solution is worst correspond with the regions of greatest deformation.

To solve this issue, different interpolants were studied. The left column of figure 5–4 shows the same simulation done using a bilinear interpolant. The middle column shows a modified bicubic interpolant where the derivatives are bounded to avoid excessive overshoots. For the top frames, the solution of the bilinear and modified bicubic interpolants is not as clean as the solution using the bicubic interpolant. But for later times (middle and bottom rows), the solutions remain stable and do not create the additional noise observed with the bicubic interpolant. Unfortunately, the linear interpolant shows a noticeable reduction of the level of details it is able to represent in the solution. The

modified bicubic seems to be a good choice as it can represent an appreciable level of details while being stable.

Other interpolants were tried. A notable one is based on a monotone cubic interpolant from [8], modified with ideas from [11]. In one dimension, these cubic interpolants are modified to make sure that no overshoot is created. However, this property is not so easy to maintain in 2D, and applying this interpolant to the two vortices simulation did not show any improvements over the regular bicubic interpolant. Many other interpolants have been studied, but none of them showed better results than the modified bicubic interpolant of figure 5–4.

5.2.4 Efficiency

As it was mentionned in chapter 4, the separation of scales made available by the diffeomorphism approach gives a considerable reduction of computational times. The advection of vorticity is computed on a coarse grid by solving equations (4.3). This step is usually expensive, especially when using a third order scheme in the gradient-augmented level set method, so solving it on a coarse grid is a great advantage. Also, the inverse Fourier transforms are performed on a coarse grid. This step is also expensive and doing it only on a coarse grid saves a lot of time.

Table 5–1 compares the computational time taken by the method depending on the grid sizes used. The fine grid is always fixed to a 512×512 grid, but the coarse grid varies from a 16×16 to a 256×256 grid. Figure 5–6 shows the result of using those different coarse grid on the solution. From top to bottom, the coarse grid has 16, 64 and 256 cells per dimension. From left to right, frames 30, 60, 90, 120, 150, 300 and 900 are shown.

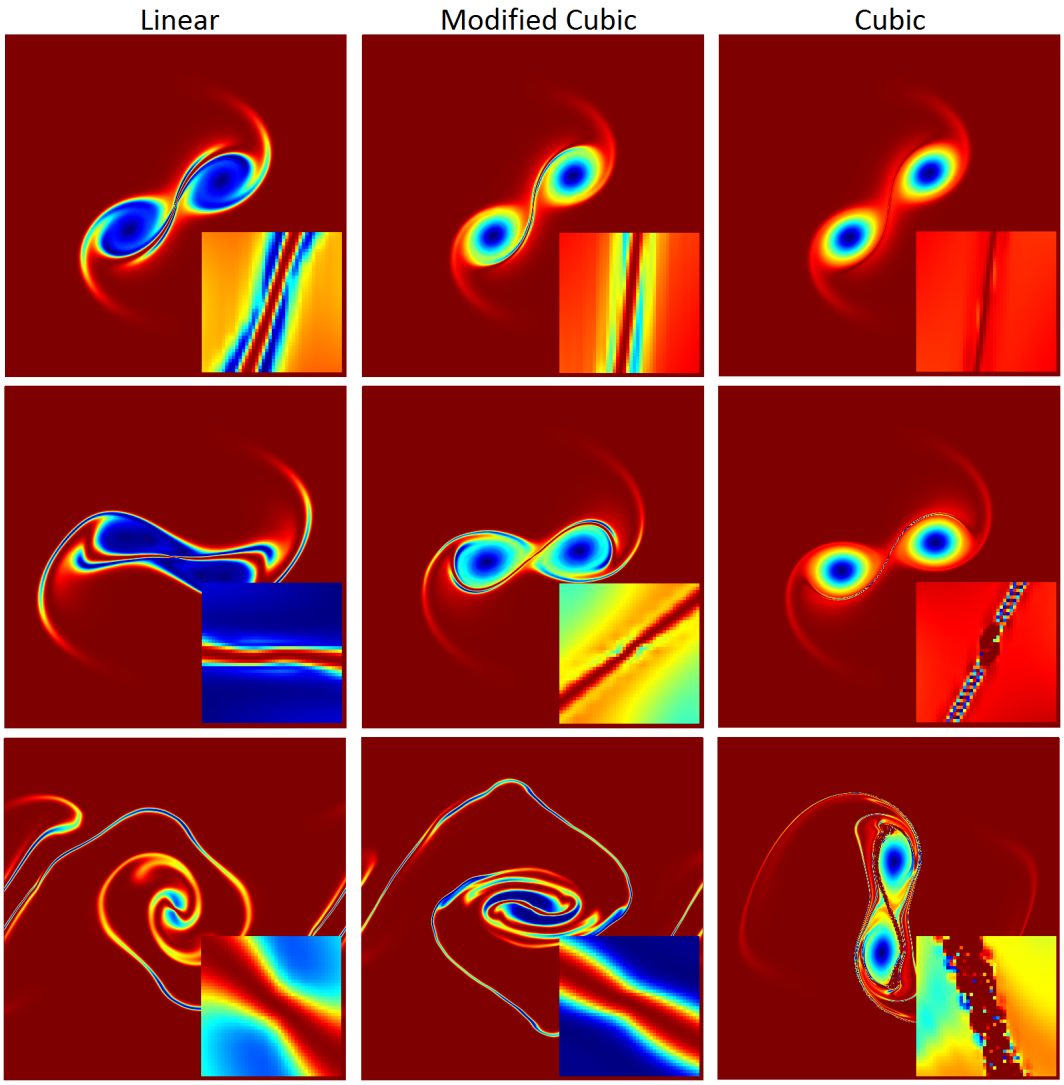


Figure 5–4: Comparison of using a bilinear, modified bicubic or bicubic interpolant to represent the solution for frames 65, 90 and 200, with a zoom of the central region. The cubic interpolant (right) shows problems in the central region due to overshoot, while the modified bicubic (middle) and bilinear (left) interpolant show a stable solution.

Coarse grid size	16	32	64	128	256
Time (sec.)	687	867	1304	2271	6034

Table 5–1: Comparison of computational times (in seconds) for different coarse grid resolutions. The fine grid is fixed at 512×512.

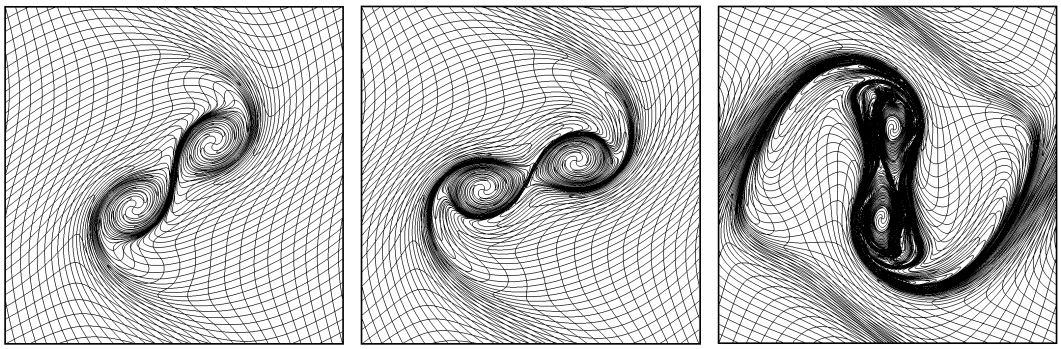


Figure 5–5: Representation of the grid transformed by the cubic diffeomorphism for frames 65, 90 and 200. The regions of greatest deformation match the representation problems seen in figure 5–4.

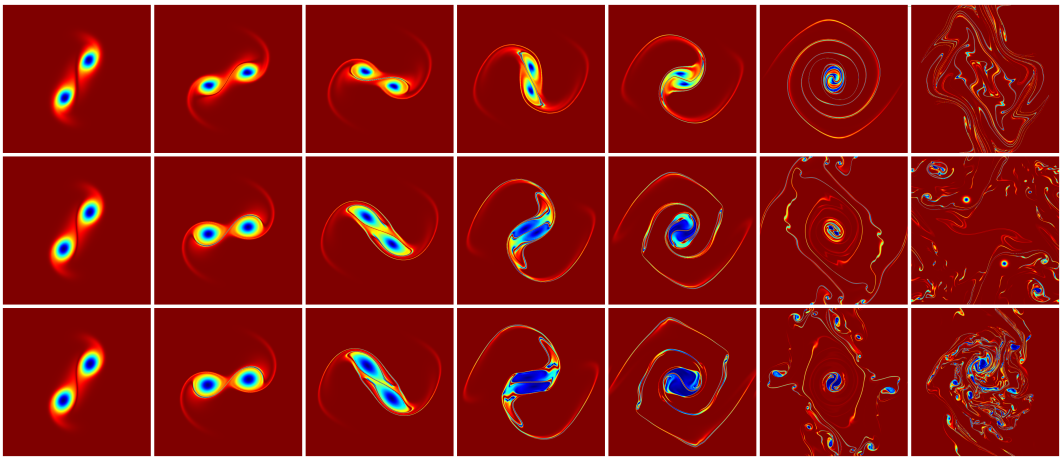


Figure 5–6: Simulation for two initial vortices with different resolutions of the coarse grid. From top to bottom, the coarse grid has 16, 64 and 256 cells per dimension. From left to right, frames 30, 60, 90, 120, 150, 300 and 900 are shown.

The reduction of the coarse grid resolution shows a significant reduction of computational times. From a 256×256 grid to a 16×16 grid, the computational time decreases by a factor 10. However, figure 5–6 shows that the solution is different depending on the grids. Using the 256×256 grid gives a more precise result and some features that should appear in the exact solution are observed, such as the *backwards cascade*, i.e. small vortices regrouping into bigger ones. This phenomenon is clearly observable in the transition between the fifth and sixth panels of the last row of figure 5–6, where thin filaments regroup to form new vortices. However, if the goal of solving the Euler equations is simply to have a visually acceptable result, then using a 16×16 or a 32×32 grid might be enough. The trade-off between speed and accuracy of the solution is, as always, dependent of the intended use of the simulation.

CHAPTER 6
Conclusion and outlook

In this thesis, we have presented several numerical methods to transport sets in a vector field. Some of those methods are well known and well tested, while others are new. We have presented in chapters 2 and 3 approaches using implicit and explicit definitions of sets, in addition to a procedure that uses diffeomorphisms in chapter 4. It is important to have a large array of methods to transports sets since every situation has its own needs, and no technique dominates all the others.

The results obtained by using our new methods in standard tests are encouraging. They show improvements over other methods with regards to accuracy and efficiency. However, the real challenge remains to couple these methods with other techniques in order to solve complex non-linear situations. We saw in chapter 5 that for a complicated system such as the Navier-Stokes equations, the advection of a set is only a part of a larger problem. We have succesfully applied the CM method of chapter 4 to the Euler equations, but using it to solve Navier-Stokes is still challenging and requires further investigation. Each physical problem has its own associated system of PDEs, sometimes with completely different properties than what we presented here, so much work remains to be done in this area.

References

[1] David Adalsteinsson and James A Sethian. A fast level set method for propagating interfaces. *Journal of Computational Physics*, 118(2):269–277, 1995.

[2] Yann Brenier. Topics on hydrodynamics and volume preserving maps. *Handbook of mathematical fluid dynamics*, 2:55–86, 2003.

[3] Faires Burden and J Faires. Reynolds. *Numerical Analysis, Prindle, Weber, & Schmidt*, 1981.

[4] Prince Chidyagwai, Jean-Christophe Nave, Rodolfo Ruben Rosales, and Benjamin Seibold. A comparative study of the efficiency of jet schemes. *International Journal of Numerical Analysis and Modeling - Series B*, 3(3):297–306, 2012.

[5] Richard Courant, Eugene Isaacson, and Mina Rees. On the solution of nonlinear hyperbolic differential equations by finite differences. *Communications on Pure and Applied Mathematics*, 5(3):243–255, 1952.

[6] Douglas Enright, Ronald Fedkiw, Joel Ferziger, and Ian Mitchell. A hybrid particle level set method for improved interface capturing. *Journal of Computational Physics*, 183(1):83–116, 2002.

[7] Lawrence C. Evans. *Partial Differential Equations*. AMS, 2010.

[8] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM, 2001.

[9] Ronald P Fedkiw, Tariq Aslam, Barry Merriman, and Stanley Osher. A non-oscillatory eulerian approach to interfaces in multimaterial flows (the ghost fluid method). *Journal of Computational Physics*, 152(2):457–492, 1999.

[10] Matteo Frigo and Steven G Johnson. Fftw: An adaptive software architecture for the fft. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 3, pages 1381–1384. IEEE, 1998.

[11] Frederick N Fritsch and Ralph E Carlson. Monotone piecewise cubic interpolation. *SIAM Journal on Numerical Analysis*, 17(2):238–246, 1980.

[12] Ami Harten and Stanley Osher. Uniformly high-order accurate nonoscillatory schemes. i. *SIAM Journal on Numerical Analysis*, 24:279, 1987.

[13] Ken Kamrin, Chris H Rycroft, and Jean-Christophe Nave. Reference map technique for finite-strain elasticity and fluid–solid interaction. *Journal of the Mechanics and Physics of Solids*, 2012.

[14] Myungjoo Kang, Ronald P Fedkiw, and Xu-Dong Liu. A boundary condition capturing method for multiphase incompressible flow. *Journal of Scientific Computing*, 15(3):323–360, 2000.

[15] Haruhiko Kohno and Jean-Christophe Nave. A new method for the level set equation using a hierarchical-gradient truncation and remapping technique. *Computer Physics Communications*, 2013.

[16] Peter Lancaster and Kes Salkauskas. Surfaces generated by moving least squares methods. *Mathematics of computation*, 37(155):141–158, 1981.

[17] Randall J LeVeque. High-resolution conservative algorithms for advection in incompressible flow. *SIAM Journal on Numerical Analysis*, 33(2):627–665, 1996.

[18] Xu-Dong Liu, Stanley Osher, and Tony Chan. Weighted essentially nonoscillatory schemes. *Journal of computational physics*, 115(1):200–212, 1994.

[19] Jean-Christophe Nave, Rodolfo Ruben Rosales, and Benjamin Seibold. A gradient-augmented level set method with an optimally local, coherent advection scheme. *Journal of Computational Physics*, 229(10):3802–3827, 2010.

[20] Stanley Osher and James A Sethian. Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations. *Journal of computational physics*, 79(1):12–49, 1988.

[21] J-P Pons, G Hermosillo, R Keriven, and O Faugeras. Maintaining the point correspondence in the level set framework. *Journal of Computational Physics*, 220(1):339–354, 2006.

[22] Benjamin Seibold. *M-matrices in meshless finite difference methods.*

[23] Benjamin Seibold, Jean-Christophe Nave, and Rodolfo Ruben Rosales. Jet schemes for advection problems. *Discrete and Continuous Dynamical Systems - Series B*, 17(4):1229–1259, 2012.

[24] James A Sethian and Alexander Vladimirsky. Fast methods for the eikonal and related hamilton–jacobi equations on unstructured meshes. *Proceedings of the National Academy of Sciences*, 97(11):5699–5703, 2000.

[25] Gunilla Sköllermo. A fourier method for the numerical solution of poissons equation. *Mathematics of Computation*, 29(131):697–711, 1975.

[26] Jos Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128. ACM Press/Addison-Wesley Publishing Co., 1999.

[27] Mark Sussman, Emad Fatemi, Peter Smereka, and Stanley Osher. An improved level set method for incompressible two-phase flows. *Computers & Fluids*, 27(5):663–680, 1998.

[28] Steven T Zalesak. Fully multidimensional flux-corrected transport algorithms for fluids. *Journal of computational physics*, 31(3):335–362, 1979.