ESAIM: PROCEEDINGS, October 2011, Vol. 32, p. 103-117
E. Cancès, N. Crouseilles, H. Guillard, B. Nkonga, and E. Sonnendrücker, Editors

SOLVING THE VLASOV EQUATION IN COMPLEX GEOMETRIES

J. Abiteboul[1], G. Latu[1], V. Grandgirard[1], A. Ratnani[2], E. Sonnendrücker[2] and A. Strugarek[1]

Abstract. This paper introduces an isoparametric analysis to solve the Vlasov equation with a semi-Lagrangian scheme. A Vlasov-Poisson problem modeling a heavy ion beam in an axisymmetric configuration is considered. Numerical experiments are conducted on computational meshes targeting different geometries. The impact of the computational grid on the accuracy and the computational cost are shown. The use of analytical mapping or Bézier patches does not induce a too large computational overhead and is quite accurate. This approach successfully couples an isoparametric analysis with a semi-Lagrangian scheme, and we expect to apply it to a gyrokinetic Vlasov solver.

Résumé. Nous présentons ici une analyse isoparamétrique pour résoudre l'équation de Vlasov à l'aide d'un schéma Semi-Lagrangien. Le cas test d'un faisceau axisymétrique d'ions lourds est étudié dans le cadre du système Vlasov-Poisson. Des tests numériques sont effectués sur différents maillages afin d'étudier diverses géométries. L'impact du choix de maillage sur la précision numérique et le coût de calcul est quantifié. L'utilisation de *mapping* analytique ou de *patches* de Bézier ne semble pas trop coûteux et permet une précision numérique suffisante. Le couplage de l'analyse isoparamétrique au schéma Semi-Lagrangien est donc réussi, nous espérons pouvoir appliquer cette méthode à des solveurs de l'équation de Vlasov gyrocinétique.

Introduction

A tokamak is a toroidal device for plasma confinement in which a strong toroidal magnetic field is imposed by a system of external coils, while a poloidal magnetic field is generated by a strong toroidal current flowing through the plasma. The sum of these toroidal and poloidal fields results in a helical geometry of the magnetic field lines. In the region we are interested in, which is the confined plasma in the core of the tokamak, the magnetic field lines are closed and are wrapped around closed surfaces. These so-called magnetic surfaces are nested around the plasma center and can usually be considered axisymmetric and described by a constant section around the torus. The shape of the magnetic surfaces has a strong impact on the physics involved in the confinement of the plasma [1]. Therefore, in order to accurately describe the transport processes in a tokamak plasma, a fine description of the geometry of magnetic surfaces is mandatory.

The strong magnetic field in a tokamak means that the motion of particles will be restricted in the direction perpendicular to the magnetic field lines, but free along this (so-called parallel) direction. Moreover, the large perturbations along the magnetic field lines play an important role in transport processes and must be included in the model. Thus, it appears that even small discretization errors can corrupt numerical results, for instance by causing a parallel heat flux to leak into the transverse direction [14]. These issues are particularly critical

[1] CEA, IRFM, F-13108 Saint-Paul-lez-Durance, France.
[2] IRMA, Université de Strasbourg and INRIA-Nancy-Grand Est, CALVI Project-Team

Article published online by EDP Sciences and available at http://www.esaim-proc.org or http://dx.doi.org/10.1051/proc/2011015

when modeling nonlinear phenomena such as turbulence, which is usually studied in the fusion community through the development of gyrokinetic codes [12]. These first-principle codes solve a coupled Vlasov-Poisson system in 5 dimensions (3 dimensions in real space and 2 in velocity space). As a first approach, most of these codes adopted a simplified description of the geometry of magnetic surfaces, for instance using basic polar coordinates, which implies that realistic tokamak geometries were not taken into account. More recently, an ongoing effort has been initiated to include more realistic geometries. As a long-term objective, we expect to design a new Vlasov solver for the gyrokinetic code GYSELA [16] which would extend the code's capability to perform simulations in complex tokamak geometry while providing a fine description of this geometry.

Classical FEM method with straight lines describing the edges are not adapted to the complex geometry of magnetic surfaces in a tokamak, where curved elements appear necessary for an accurate description of these surfaces. In this context, the isoparametric and isogeometric frameworks provide a method to produce curved elements adapted to any given geometry, and to create a PDE solver using these elements. Compared to a more standard description by metric tensors, the isoparametric/isogeometric frameworks: 1) give more flexibility to generate and refine the computational mesh, 2) introduce a rich set of computationally cheap parametric functions to build the mesh and to map quantities between configuration space and parameter space, 3) provide the so called k-refinement, a strategy that can increase the regularity of functions through the mesh's interfaces, reducing the number of degrees of freedom, 4) allow a simple modification of the boundary by repositioning the control points.

In order to advance towards the long-term objective of designing a gyrokinetic Vlasov solver in complex geometry, we describe in this paper a feasibility study of isoparametric analysis in a simplified setting. The Vlasov solver is implemented on a reduced phase space (1D in space and 1D in velocity), instead of the 5D phase space of gyrokinetic codes. This reduced model is directly relevant for the application expected in GYSELA, as isoparametric meshes would be used – at least as a first step – to generate the 2D meshes describing magnetic surfaces in the poloidal plane. We consider a Vlasov-Poisson problem modeling the focusing of a heavy ion beam in an axisymmetric geometry around its optical axis. The advantage of this reduced model is that numerical experiments can be performed in a small-sized phase space domain using simple Dirichlet boundary conditions. Since the solution remains very localized, one can easily investigate different mesh geometries without damaging numerical results. Moreover, as is the case in gyrokinetic simulations, small-scale filamentation develops in phase-space, which must be correctly captured by the solver independently of the chosen mesh. The aim of this paper is to validate numerically the Vlasov solver for different underlying meshes, as well as to measure the impact of the chosen mesh on computational costs.

In section 1, we describe the two-dimensional Vlasov-Poisson test case considered, as well as the numerical methods used for a standard cartesian mesh. The framework for isogeometric analysis and the chosen computational grids are introduced in section 2. The specific algorithms needed to design a Vlasov solver on an isoparametric surface are presented in section 3. The results in terms of accuracy and computational cost for the different geometries considered are detailed in section 4. A conclusion follows.

1. 2D Vlasov

1.1. **Physical model: the paraxial beam**

Our reference test case is the study of beam focusing using the paraxial Vlasov-Poisson model [10]. This model is common in accelerator physics and describes the propagation of a particle beam along a linear optical axis. While the beam is considered in steady-state, the propagation velocity in the direction z of the optical axis is assumed constant (thus z is formally replaced by time in the equations). We also assume that the beam is symmetric around its optical axis. Therefore, we solve the Vlasov-Poisson system in cylindrical geometry with the radius r as the only space coordinate and v_r the velocity in the radial direction. In order to avoid issues with boundary conditions around $r = 0$, we consider a symmetric domain in r with the condition $f(-r, v_r) = f(r, -v_r)$ where f is the particle distribution function in phase-space.

Following the normalization in [25], we solve the Vlasov equation coupled with a Poisson equation

$$\partial_t f + v_r \partial_r f + F(t,r)\partial_v f(t,r,v_r) = 0 \tag{1}$$

$$\frac{1}{r}\partial_r \left(r E_{self}(t,r)\right) = \rho(t,r) = \int_{-\infty}^{\infty} f(t,r,v_r)\,\mathrm{d}v_r \tag{2}$$

where $F(t,r)$ is the applied force. It contains contributions from both the externally applied (E_{app}) and the self-applied (E_{self}) fields:

$$F(t,r) = K_{self}\, E_{self}(t,r) + K_{app}\, E_{app}(t,r)\ .$$

where K_{app} and K_{self} are constants. We consider the case where a proton beam is focused by applying a periodic external electric field with the following normalized expression

$$E_{app}(t,r) = -\frac{1}{2}\left\{1+\cos\left(2\pi t\right)\right\}^2 r \tag{3}$$

As an initial distribution function, we will use the semi-gaussian (gaussian in velocity, localized in space) formulation

$$f_0(r,v_r) = \frac{N_0}{\sqrt{2\pi}\pi a^2 v_{th}} e^{-\frac{v_r^2}{2v_{th}^2}} \quad \text{for} \quad |r| < a \tag{4}$$

$$= 0 \qquad\qquad\qquad\qquad \text{elsewhere} \tag{5}$$

where N_0 is the total number of particles, a is the initial beam radius and v_{th} is the thermal velocity.

1.2. The ISOLOSS code

In order to solve the 2D Vlasov-Poisson system in complex geometry, we have used the LOSS code [7], extending it for the paraxial beam test case and for various geometries to a new version named ISOLOSS. The ISOLOSS code uses a semi-Lagrangian numerical scheme [26], which will be presented in section 1.2.1. The time integration is performed using a predictor-corrector scheme, described in section 1.2.2.

1.2.1. *Semi-Lagrangian method*

Two types of methods are most commonly used to solve the Vlasov-Poisson system. On the one hand, Lagrangian (or Particle-In-Cell) methods [4] discretize the distribution function into a finite number of macro-particles. The evolution of these macro-particles then follows the characteristics of the Vlasov equation, while a grid is necessary only in real space in order to solve the Poisson equation. Lagrangian methods allow for efficient computations with a numerical cost that can be tuned depending on the expected accuracy. The main drawback of such methods is the numerical noise due to the sampling of the distribution function, which requires complex noise reduction methods (see for instance [19] for state-of-the-art noise reduction techniques in gyrokinetic simulations). On the other hand, Eulerian methods solve the Vlasov equation on a fixed grid in phase-space using finite differences, finite volumes or spectral methods to discretize the operators in the Vlasov equation. The key issue for Eulerian methods is that the discretization of the operators leads to numerical dissipation.

The semi-Lagrangian method is a mix between the Lagrangian and Eulerian methods, which tries to eliminate the main drawbacks of each method. This method was first developed for meteorological studies (see [27] for a review) and was more recently adapted to plasma simulations [26]. In order to avoid the statistical noise observed in Lagrangian codes, a fixed (Eulerian) grid in phase-space is used. On the other hand, to avoid numerical dissipation and take advantage of the conservation of the distribution function along trajectories by the Vlasov equation, the characteristics of the equation are used to compute the time evolution of the distribution function. The basic algorithm for the backward semi-Lagrangian method [26] used in LOSS is described in figure 1. For every grid point at a given time step, the characteristic curves are integrated backward to find the value

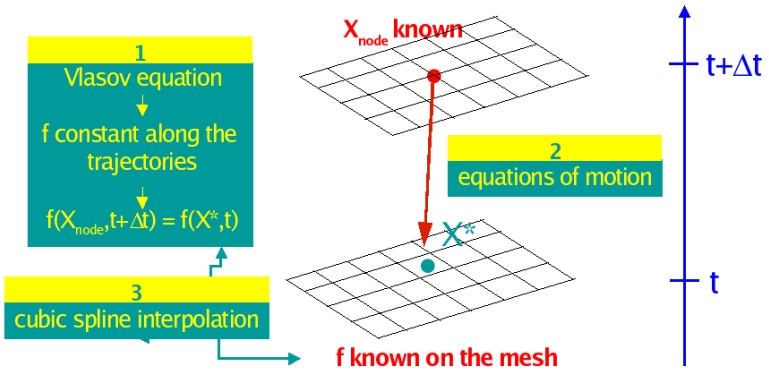


FIGURE 1. Basic algorithm for the Semi-Lagrangian method (figure from [15])

of the distribution function at the foot of the characteristic. As this point is hardly ever on the grid, an interpolation must be performed to compute the value of the distribution function. It has been shown [3, 11] that cubic spline interpolation provides a good compromise between accuracy (low diffusivity) and numerical cost. Details of this method will be presented in section 1.2.4.

1.2.2. *Predictor-corrector scheme*

The time integration of the Vlasov-Poisson system is performed using a second-order in time predictor-corrector numerical scheme. Given the distribution function $f(t^n, r, v_r)$ at time t^n we perform the following sequence to compute f at $t^{n+1} = t^n + \Delta t$:

-1- Computation at time t^n of the charge density ρ^n (different methods are considered for this computation, depending on the choice of geometry, see 3.3 for details)
-2- Computation of the self-consistent electric field E^n by solving the Poisson equation (2) with $\rho = \rho^n$
-3- For each grid point (r, v_r): backward advection of $\Delta t/2$ to compute the foot of the characteristic (r^*, v_r^*)
-4- Interpolation on the 2D grid using local cubic splines to compute $f(t^n, r^*, v_r^*)$, which yields the value for $\tilde{f}(t^{n+1/2}, r, v_r)$ due to the conservation of the distribution function along the characteristics
-5- Computation of the density $\tilde{\rho}^{n+1/2}$ and electric field $\tilde{E}^{n+1/2}$
-6- For each grid point (r, v_r): backward advection (and interpolation) of Δt using the fields at $t^{n+1/2}$ to compute $f(t^{n+1}, r, v_r)$ from the values $f(t^n)$

This numerical scheme is similar to a second-order Runge-Kutta method.

1.2.3. *Following the characteristics*

In the semi-Lagrangian method, one has to compute characteristic curves between two consecutive time steps. The foot of one characteristic, denoted previously (r^*, v_r^*), is found by approximating an advection term using the velocity and the force fields. Several procedures can be used in order to find this foot, such as Taylor expansion [15] or fixed-point iterations. We choose a predictor-corrector scheme that gives an advection term approximated with an error of second order in space (see [6] for detailed description).

1.2.4. *2D cubic spline interpolation*

The interpolation of the distribution function on the grid in phase-space is one of the main challenges of the semi-Lagrangian method. The method used for this interpolation should limit dissipation while being numerically efficient, as it will be performed at each time step and for every grid point. In the LOSS and ISOLOSS codes, this interpolation is performed using two-dimensional cubic splines [8, 26]. In this section, we will first present the method for interpolation with cubic splines in one dimension, then extend this method to two dimensions using a tensor product of cubic spline bases.

In one dimension, consider a function $g(x)$ defined for $x \in [x_0, x_N]$. This function g is projected on a basis of cubic splines:

$$g(x) \simeq c_s(x) = \sum_{\nu=-1}^{N+1} \eta_\nu B_\nu(x) \tag{6}$$

where B_ν are the cubic B-splines and the coefficients η_ν are the unknown spline coefficients. The interpolation of the function g by c_s on the $N+1$ points of the domain leads to $N+1$ equations ($c_s(x_i) = g(x_i)$ for $i = 0, \ldots, N$) for the $N+3$ spline coefficients. As the linear system is undetermined, we add two constraints on the derivatives at the boundaries

$$g'(x_i) \simeq {c_s}'(x_i) = \frac{1}{2h}\left(\eta_{i+1} - \eta_{i-1}\right) \quad \text{for} \quad i = 0 \text{ and } i = N \tag{7}$$

where h is the uniform cell size defined by $h = x_{i+1} - x_i$. Thus the spline c_s interpolating g at the grid points and verifying the boundary conditions is uniquely defined. The vector of spline coefficients $\eta = [\eta_{-1}, \ldots, \eta_{N+1}]^T$ is obtained by solving the linear system $A\eta = G$ where

$$G = [g'(x_0), g(x_0), \ldots, g(x_N), g'(x_N)]^T \tag{8}$$

and A is the following $(N+3) \times (N+3)$ matrix:

$$A = \frac{1}{6}\begin{pmatrix} -3/h & 0 & 3/h & 0 & \cdots & 0 \\ 1 & 4 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & & 1 & 4 & 1 \\ 0 & \cdots & 0 & -3/h & 0 & 3/h \end{pmatrix} \tag{9}$$

Note that the Hermite boundary conditions can, depending on the simulation domain, be replaced by periodic boundary conditions. It modifies the linear system that needs to be solved but not the general method. The LU decomposition of matrix A is easily computed using Gauss elimination. For a given vector G, the spline coefficients are then computed in two steps by solving successively the lower triangular matrix system $L\mu = G$ and the upper triangular matrix system $U\eta = \mu$.

Extending this method from one dimension to two dimensions is achieved by considering the tensor product of two cubic spline bases. Consider a function $g(x, y)$ defined for $(x, y) \in [x_0, x_{N_x}] \times \left[y_0, y_{N_y}\right]$. The interpolating spline becomes:

$$g(x, y) \simeq c_s(x, y) = \sum_{\nu=-1}^{N_x+1} \sum_{\beta=-1}^{N_y+1} \eta_{\nu,\beta} B_\nu(x) B_\beta(y) \tag{10}$$

where the unknowns are the $(N_x + 3) \times (N_y + 3)$ coefficients $\eta_{\nu,\beta}$. We first solve the following system for each value of $j = 0, \ldots, N_y$

$$c_s(x, y_j) = \sum_{\nu=-1}^{N_x+1} \gamma_\nu(y_j) B_\nu(x) \quad \text{where} \quad \gamma_\nu(y_j) \equiv \sum_{\beta=-1}^{N_y+1} \eta_{\nu,\beta} B_\beta(y) \tag{11}$$

For each value of j, we obtain $N_x + 1$ interpolation conditions. Imposing values of the derivatives at the boundaries in x leads to $N_y + 1$ linear systems of the type $A\gamma_\nu(y_j) = G$ where A is the matrix (9) and the vector G is similar to expression (8). These $N_y + 1$ systems of size $(N_x + 3) \times (N_x + 3)$ are solved using LU decomposition as described for one dimensional interpolation, yielding the solutions $\gamma_\nu(y_j)$ for $\nu = -1, \ldots, N_x+1$ and $j = 0, \ldots, N_y$.

To compute the coefficients $\eta_{\nu,\beta}$ in equation (10), we still need to solve equation (11) for $\nu = -1, \ldots, N_x + 1$. For a given value of ν, the systems previously solved give us $N_y + 1$ interpolation conditions, while we need

to solve for $N_y + 3$ unknowns. We force the derivatives at the boundaries in y to complete the system, which implies that we need to compute $\gamma'_\nu(y_0)$ and $\gamma'_\nu(y_{N_y})$. This means solving two additional linear systems of the type $A\gamma'_\nu(y_j) = \partial_y G$ with $j = 0$ and $j = N_y$, where the vector G is defined by derivatives of $g(x, y)$ and A is the matrix in (9). Finally, for each value of ν, we solve the system $A\eta_{\nu,\beta} = \Gamma_{\nu,\beta}$ with $\Gamma_{\nu,\beta} = \left[\gamma'_\nu(y_0), \gamma_\nu(y_0), \ldots, \gamma_\nu(y_{N_y}), \gamma'_\nu(y_{N_y})\right]^T$, once again using LU decomposition to obtain the spline coefficients $\eta_{\nu,\beta}$.

To estimate the computational cost of two-dimensional cubic spline interpolations, we need to count the number and size of the linear systems solved by one LU decomposition. To compute all the coefficients, we need to solve:

- $N_y + 1$ systems of size $(N_x + 3) \times (N_x + 3)$
- 2 systems of size $(N_x + 3) \times (N_x + 3)$
- $N_x + 3$ systems of size $(N_y + 3) \times (N_y + 3)$

Considering that, since the matrix A is almost tridiagonal, the resolution through LU decomposition of a linear system of size $N \times N$ requires $\mathcal{O}(N)$ operations, the global cost of two-dimensional cubic spline interpolation is of the order $\mathcal{O}(N_x N_y)$ operations. The cubic B-splines have compact support basis, the 1D interpolation at a given position requires only the combination of 4 coefficients with 4 basis functions. Let us assume a uniform grid spacing Δx from x_{min} to x_{max}, one has

$$c_s(x) = \sum_{\nu=\lfloor \frac{x-x_{min}}{\Delta x} \rfloor - 1}^{\lfloor \frac{x-x_{min}}{\Delta x} \rfloor + 2} \eta_\nu B_\nu(x) \ . \tag{12}$$

In a two dimensional setting, the interpolation uses 16 spline coefficients combined with the evaluation of 8 basis functions (4 in x and 4 in y) at that point, leading to a reasonable cost of a few tens of floating point operations. In the sequel, $\texttt{Interp}(g, x, y)$ denotes the 2D interpolation of function g taken at position (x, y).

2. Complex geometry using parametric surfaces

In the following, the symbol t will no longer be used as the time variable but as a parameter in the spatial parametric equations.

2.1. General framework

The need of an accurate representation of the geometry is not an exclusive matter of a specific application domain but is quite common in scientific computing. *Diffeomorphisms* associated to finite elements have proved to be very useful to deal with complex geometries. The idea is to perform a shape transformation in order to map a computational domain, for example a rectangular grid, to a potentially complex geometrical domain. At a given mesh resolution, a well chosen diffeomorphism adapted to the geometry can reduce numerical errors.

Diffeomorphisms and mapping techniques provide methods to go from a reference coordinate system (*e.g.* $(s, t) \in [0, 1]^2$) to a physical coordinate system (*e.g.* $(x, y) \in \Omega$). To do so, one has to choose a *shape function* or a *mapping* to go from the reference system to the physical system. A common example is the use of a polar mapping to map a rectangular grid onto a circular domain. In our context, we choose to work on a uniform grid in the reference coordinate system. Such techniques have also been used in the context of moving grids for the semi-Lagrangian method [24].

In this context, the use of B-splines and NURBS may provide a simple and powerful tool. Since their introduction in the 1960s, B-splines (then NURBS) were almost exclusively used in the CAD community. Recently, Hughes [17] has made the link between the CAD community and the simulation one. The isogeometric analysis gives the user the ability to approach numerical solutions of partial differential equations in the same space used to describe the domain. Isoparametric analysis is generally attributed to Taig [28] and Irons [18].

The idea is to approximate the domain using shape functions. Those functions are then used to approximate the solution of partial differential equations. In fact, when one uses high-order elements for solving a partial differential equation in a domain with curved boundaries, it is essential to include some way of approximating the boundary conditions accurately. The use of isoparametric elements is an efficient way which involves a general piecewise-polynomial change of variables $\mathbf{F}$ in the definition of the discrete spaces. Recalling the classical formulation of the FEM, we use an affine transformation to map a given element K_e to the reference element K. The element basis functions φ_e in K_e are then related to the reference basis functions φ by the relation $\varphi_e(\mathbf{x}) = \varphi(\mathbf{F}^{-1}(\mathbf{x}))$. Elements where the mapping $\mathbf{F}$ comes from the same finite element space are called *isoparametric*. One cost to pay is that this chain rule will be needed for each derivative evaluation. Notice that we must be careful to obtain mappings that have regular jacobians [5, 20]. When the shape functions are B-splines or NURBS, and when the domain is exactly modeled using those shape functions, Isoparametric analysis becomes Isogeometric analysis. Recall that we can model exactly all conics using NURBS.

One of the most interesting aspects of B-splines is that once the domain is described by B-splines, several strategies exist to refine the mesh:

- by inserting new knots. This is the h-refinement, it is the equivalent of mesh refinement of the classical finite element method,
- by elevating the B-spline degree. This is the p-refinement, it is the equivalent of using higher finite element order in the classical FEM,
- by increasing / decreasing the multiplicity of inserted knots. This is the k-refinement. This new strategy does not have any equivalent in the classical FEM.

One of the most interesting points of such mappings is that the evaluation for a particular point depends only on a finite (and quite small, depending on the spline degree) number of data, which makes the algorithms strongly local. In figure 2, one can see that a curve lying between two (interpolating) controls points is uniquely determined by the control points defining the corresponding control polygon. This local dependence is very useful as one can change the shape of a curve by only changing the position of some control points (see figure 2).

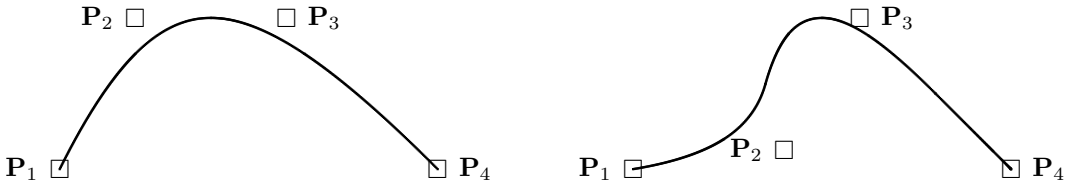


FIGURE 2. A Bézier curve after repositioning the control point $\mathbf{P}_2$

The inconvenient of B-spline mappings is the inverse problem: given a point in the physical domain, what is the corresponding parametric point? Iterative algorithms to perform this inverse mapping will be described in section 3.1. Another idea is to use an analytical inverse, whenever it is possible, for example in the case of a polar grid (see section 2.2).

A main idea included in this tool set, is to be able to deal with some class of spatial domains, described by B-splines. The aforementioned description can be coupled with a Semi-Lagrangian method to solve the Vlasov equation. The resulting code, that we have named ISOLOSS, will be described in the sequel of this paper. The present work focuses on two isoparametric meshes: an oval mesh with an analytical inverse mapping (section 2.2) and a mesh defined by Bézier elements (section 2.3), which are a specific type of NURBS or B-splines.

2.2. **Analytical mapping**

As a first step, we consider the case where the mapping is performed by a fully analytical diffeomorphism. This diffeomorphism maps a rectangle $(s,t) \in [0,1] \times [0, 2\pi[$ (in the reference space) to an ellipse in the physical space. The following parametric equations are taken as a mapping function:

$$x(s,t) = r_{max} \times s \times \cos(t) \qquad y(s,t) = v_{max} \times s \times \sin(t)$$

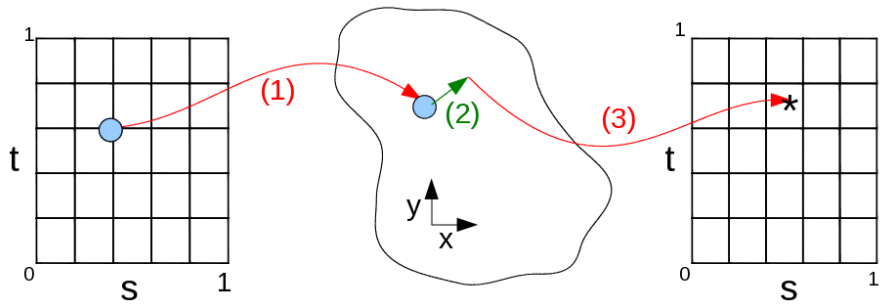


FIGURE 3. The Semi-Lagrangian method in complex geometry: (1) map the position in the reference space into physical space, (2) follow the characteristic backwards in physical space, (3) map the obtained position back in the reference space to perform the interpolation

The inverse mapping is also given by analytical expressions:

$$s(x,y) = \sqrt{\frac{y^2}{v_{max}^2} + \frac{x^2}{r_{max}^2}} \qquad t(x,y) = \begin{cases} \arctan(\frac{y}{x}) & \text{if } x > 0 \text{ and } y \geq 0 \\ \arctan(\frac{y}{x}) + 2\pi & \text{if } x > 0 \text{ and } y < 0 \\ \arctan(\frac{y}{x}) + \pi & \text{if } x < 0 \\ \frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0 \\ \frac{3\pi}{2} & \text{if } x = 0 \text{ and } y < 0 \end{cases}$$

The computational cost of moving forward and backward from the reference space to the physical space is quite low. A direct or inverse mapping represents a fraction of the cost induced by one 2D interpolation. Therefore, as we shall see in the numerical experiments, the overhead in the ISOLOSS code is small compared to the original solution that does not require a diffeomorphism.

2.3. **Bézier patches**

A Bézier surface (also known as Bézier patch) of order (n, m) is defined by several control points $(\mathbf{k}_{i,j})_{i\in[0,n],j\in[0,m]}$. It maps the unit square $[0,1]^2$ into a surface embedded within a space of the same dimensionality as the control points. In our application, we take the control points $\mathbf{k}_{i,j}$ in $\mathbb{R}^2$ in order to represent a 2D geometry. A Bézier surface is parametric and the equations describing it depend on parameters that are not explicitly part of the geometry. Hence, a point P of coordinates (s,t) on the patch is localized at the following position in the physical space:

$$\mathbf{P}(s,t) = \sum_{i=0}^{n}\sum_{j=0}^{m} \mathcal{B}_i^n(s)\mathcal{B}_j^m(t)\ \mathbf{k}_{i,j} \qquad \text{with} \qquad \mathcal{B}_i^n(u) = \binom{n}{i} u^i(1-u)^{n-i}$$

$\mathcal{B}_i^n(u)$ are known as the Bernstein basis of polynomials of degree n. As a first step, we decide to use biquadratic Bézier patches.

3. ALGORITHMS

The Semi-Lagrangian method, as described in section 1.2, must be adapted to the formalism of parametric surfaces introduced in section 2. In the approach proposed in this paper, the choice has been made to keep the expressions of the advection equations in the physical space, rather than rewriting these equations in the reference space [2]. Therefore, it is necessary to move backward and forward between the reference space and the physical space. Figure 3 outlines the adapted Semi-Lagrangian method when using parametric surfaces. From a given position (s,t) in the reference space, we map the corresponding position in physical space. The characteristics are then followed backwards in physical space, and the foot of the characteristics must be mapped

back into the reference space before performing the cubic spline interpolation. The main issue in this algorithm is the inverse mapping of positions from physical space to reference space. This transformation has to be performed for each grid point during the advection step. There is no generic analytical solution for curved elements and this operation can be numerically costly. Several solutions have been proposed in the computer graphics community to design fast inverse mappings [13, 22]. Algorithms to perform the inverse mapping from physical space to reference space are presented in section 3.1.

Another issue when solving the Vlasov-Poisson system on an isoparametric mesh is the computation of velocity integrals to obtain the density, which is needed to solve the Poisson equation. Whereas such computation is trivial for a cartesian grid in position and velocity, it requires a special treatment for curved elements, which will be presented in section 3.3.

3.1. **Inverse mapping for Bézier patches**

3.1.1. *Existing solutions*

Several algorithms have been proposed to calculate the inverse mapping for Bézier patches [13]. A common approach is to use a multivariate Newton iteration scheme (also called Newton-Raphson scheme). This method uses an iterative process to approach one root of a set of equations. For an adequately suited set of equations, this procedure starts with the position in physical space and gives the position in the reference space within a few iterations. The convergence is quadratic and the number of significant digits doubles after each iteration. The Newton scheme has an intrinsic problem: it may fail to converge. As the convergence depends on the gradient, we can expect Newton algorithms to fail for domains with singularities.

A method called Bézier clipping [22] has been introduced to guarantee systematic convergence. Iteratively, the algorithm builds smaller nested Bézier subpatches. These Bézier subpatches keep the target point inside them. The parameter domain that surrounds the target point is thus iteratively reduced until it becomes sufficiently small that one can approximate easily the inverse mapping in averaging the control points of the subpatch. This strategy involves much more computations than the Newton-Raphson method.

The inverse mapping algorithm we need is based on the same techniques used for computing the points at which a ray intersects a rational Bézier patch in the computer graphics community. In this work we only investigate Newton and clipping schemes. Some alternative methods (such as interval Newton iteration) have been described in [29].

The inverse mapping function used in the sequel is denoted $\texttt{Imap}(r, v_r)$. If the input point (r, v_r) is in the computational domain then one has $\texttt{Imap}(r, v_r) = (s, t) \in [0, 1]^2$.

3.1.2. *Description of the Newton algorithm*

The inverse mapping problem can be written as:

$$\mathbf{P}(s^\star, t^\star) = \sum_{i=0}^{n} \sum_{j=0}^{m} \mathcal{B}_i^n(s^\star)\mathcal{B}_j^m(t^\star)\, \mathbf{k}_{i,j} = \begin{pmatrix} x^\star \\ y^\star \end{pmatrix}$$

where $s^\star$ and $t^\star$ are the unknown parameters in reference space. We apply Newton's method to solve this nonlinear system. The equation can be rewritten as

$$f(s^\star, t^\star) = 0 \quad \text{with} \quad f(s,t) = P(s,t) - \begin{pmatrix} x^\star \\ y^\star \end{pmatrix}$$

The Taylor series of $f(s,t)$ around the point $f(s^\star, t^\star)$ where f is equal to zero is given by

$$f(s^\star, t^\star) = f(s,t) + J(s,t) \begin{pmatrix} s^\star - s \\ t^\star - t \end{pmatrix} + \ldots, \quad \text{with} \quad J = \begin{pmatrix} \frac{\partial P_s}{\partial s} & \frac{\partial P_s}{\partial t} \\ \frac{\partial P_t}{\partial s} & \frac{\partial P_t}{\partial t} \end{pmatrix}$$

At first order we get $\begin{pmatrix} s^\star \\ t^\star \end{pmatrix} = \begin{pmatrix} s \\ t \end{pmatrix} - J^{-1}(s,t)[f(s^\star,t^\star) - f(s,t)]$.

The Newton iteration is then $\begin{pmatrix} s^{k+1} \\ t^{k+1} \end{pmatrix} = \begin{pmatrix} s^k \\ t^k \end{pmatrix} - J^{-1}(s^k,t^k)[f(s^{k+1},t^{k+1}) - f(s^k,t^k)]$.

The inverse of the Jacobian matrix J can be accurately computed in the Bézier formulation, using derivatives of the Bernstein polynomials. The iterative algorithm continues until one of the following three criteria is met:

(1) the l^2 norm of $\begin{pmatrix} s^{k+1} - s^k \\ t^{k+1} - t^k \end{pmatrix}$ is lower than a threshold,
(2) the number of iterations is larger than a predefined maximum number,
(3) the position (s,t) is no longer in the Bézier patch.

Note that, in the latter two cases where the Newton algorithm fails to converge, it is possible to switch to the clipping algorithm described in the following. The Newton algorithm needs to evaluate surface points as well as partial derivatives for given parameter values (s,t). In order to reduce the number of computations in the Newton kernel, factorizations are performed between the computations of $f(s_k,t_k)$, $\frac{\partial P}{\partial s}(s_k,t_k)$ and $\frac{\partial P}{\partial t}(s_k,t_k)$.

3.1.3. *Description of the clipping algorithm*

The clipping algorithm uses properties associated to Bézier patches:

- *Convex hull property*: a Bézier surface lies completely within the convex hull of its control points.
- *Subdivision Algorithm for Bézier Curves*: using the DeCasteljau algorithm, one can write a quick algorithm to subdivide a Bézier patch into two Bézier subpatches (see for instance [9]).

Let us describe briefly the different steps of the Bézier clipping (for more details, see [21–23, 30]). As an input, we have a Bézier patch with its control points $\mathtt{C}_{i,j}$ (position in parameter space $(s_{i,j},t_{i,j})$, position in space $(x_{i,j},y_{i,j})$). The idea is to reduce the interval of possible values of $s \in [0,1]$ to a smaller interval that contains the target point P. As a first step, one tries to approximate the intersection of the Bézier patch with a line $\mathtt{L}$ in the direction of the t-axis that goes through P. Suppose that we find a way to know that the intersection of this line $\mathtt{L}$ with the Bézier patch contains only values of s in the interval $[s_{min}, s_{max}]$, one can then deduce immediately that the s coordinate of P is also in the same interval.

To find such an interval, the convex hull property can be used as follows: the intersection of the Bézier patch with $\mathtt{L}$ has to be included in the intersection of the convex hull (defined by the control points $\mathtt{C}_{i,j}$) with $\mathtt{L}$. Computing the intersection of the convex hull with $\mathtt{L}$ is a non trivial procedure, which we present here. First, one defines a line $\mathtt{L}$ almost perpendicular to the s-axis that traverses the target point P. Then, the $d_{i,j}$ distance of each control point $\mathtt{C}_{i,j}$ to this line is computed. Because of the perpendicular property of line $\mathtt{L}$, one can expect that control points at $s=0$ will be often negative and these at $s=1$ will be often positive. A new Bézier patch is built by replacing the previous control points at $(s_{i,j},t_{i,j})$ with new control points at the coordinates $(s_{i,j},d_{i,j})$ in parameter space. This patch represents a projection that simplifies the intersection task. Suppose the convex hull of these new control points intersects the s-axis at two points s_{min} and s_{max}, such that $0 \leq s_{min} \leq s_{max} \leq 1$. Then we know the intersection points of the Bézier curve and s-axis will be inside $[s_{min}, s_{max}]$. We subdivide the Bézier patch into three patches, with s ranging in $[0, s_{min}]$, $[s_{min}, s_{max}]$, and $[s_{max}, 1]$ respectively. We know that the intervals $[0, s_{min}]$ and $[s_{max}, 1]$ can be safely discarded. Next, we can iterate the whole procedure in the other direction (t) on the new Bézier patch restricted to $s \in [s_{min}, s_{max}]$.

Finally, the algorithm loops onto the described process until the intervals for each parameter are small enough. The proof of convergence of the algorithm is based on the following fact: at each iteration one substracts parameter ranges which are guaranteed not to include the target point.

A sketch of the nested Bézier patches generated during the clipping is depicted in figure 4 for an input point located at $(2,3)$.

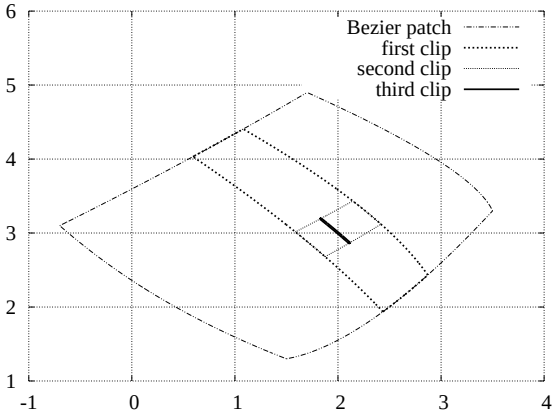


FIGURE 4. Nested Bézier patches generated during clipping algorithm

3.2. **Reducing delays for patch finding**

In order to achieve good performance, it is crucial to reduce the total number of inverse mapping calculations to the minimum. In theory, when applying the Bézier clipping algorithm for inverse mapping, we must iterate over all the Bézier patches until finding the one which actually contains the point we are trying to map. A useful strategy (common in the computer graphics community) consists in subdividing space into more manageable and smaller chunks. We choose to build a collection of axis-aligned bounding boxes around each Bézier patch. Due to the convex hull property of Bézier surfaces, the control points allow us to determine an axis-aligned bounding box in physical space for each patch. For a given point in physical space, if it does not belong to a given bounding box, then it is certain that the point is not inside the associated Bézier patch. One can take advantage of this property to restrict the search of the inverse mapping procedure to a small set of Bézier patches, typically between one and four potential patches. This strategy shortens the time needed to find the patch in which the target point lies.

In order to further improve this strategy, we perform some work in a preprocessing stage to remove costs during inverse mapping computations [21]. Initially, the entire physical domain is bounded by a global bounding box and this box is cut into voxels of equal size. The size of the voxels is chosen such that the volume of a voxel is significantly smaller than the volume of the average bounding box of a Bézier surface. Inside each voxel, a list of the Bézier patches is stored. Given an input point lying in one voxel, this list gives all Bézier patches which could encompass the input point. The list is built based on the axis-aligned bounding boxes previously computed.

Finally, the procedure adopted to obtain the inverse mapping of an input point using Bézier clipping is the following: 1) read the voxel where the list of potential Bézier patches is stored, 2) try applying the clipping algorithm inside each patch of the list and stop as soon as the input point belongs to one of the patches 3) return the localization $(s^\star, t^\star)$ given by the clipping algorithm.

In all the test cases considered in this work, where the meshes do not present any singularity, the Newton-Raphson algorithm always converges. Both methods for inverse mapping have been implemented, but the computational cost of the Bézier clipping algorithm is greater by at least one order of magnitude. Therefore, we have used the Newton-Raphson algorithm for the results presented in section 4. More generally, a good compromise can be found by using the faster Newton-Raphson algorithm, with the "safer" Bézier clipping algorithm as a backup solution when the Newton scheme does not converge, for instance when the grid presents singularities.

3.3. **Velocity integrals**

The density $\rho(r)$ must be obtained in order to solve the Poisson equation. This density is computed as an integral over the velocity direction, which is not trivial when using a curved grid in phase-space. Indeed, we need a numerical quadrature for approximating the integral over a curved domain represented by isoparametric elements. The inputs available for this computation are the values of the distribution function at the control points positions, and also the spline coefficients used for the interpolation. We choose to approximate the integrals using the trapezoidal rule coupled with a spline interpolation. Let us define the following series $r_q = r_{min} + q\Delta r$ for $q = 0, N_r - 1$ and $v_k = v_{min} + k\Delta v$ for $k = 0, N_v - 1$. The sampled ρ integrals and associated computations are

$$\begin{aligned}
\rho(r_q) &= \Delta v \sum_{k=0}^{N_v-1} f(r_q, v_k) \ , \qquad (s^\star_{q,k}\ , t^\star_{q,k}) = \texttt{Imap}(r_q, v_k) \ , \\
f(r_q, v_k) &= \texttt{Interp}(f, s^\star_{q,k}, t^\star_{q,k}) = \sum_{\nu=\lfloor \frac{s^\star_{q,k}}{\Delta s} \rfloor - 1}^{\lfloor \frac{s^\star_{q,k}}{\Delta s} \rfloor + 2} \ \sum_{\beta=\lfloor \frac{t^\star_{q,k}}{\Delta t} \rfloor - 1}^{\lfloor \frac{t^\star_{q,k}}{\Delta t} \rfloor + 2} \eta_{\nu,\beta} B_\nu(s^\star_{q,k}) B_\beta(t^\star_{q,k}) \ , \\
\rho(r_q) &= \Delta v \sum_{k=0}^{N_v-1} \ \sum_{\nu=\lfloor \frac{s^\star_{q,k}}{\Delta s} \rfloor - 1}^{\lfloor \frac{s^\star_{q,k}}{\Delta s} \rfloor + 2} \ \sum_{\beta=\lfloor \frac{t^\star_{q,k}}{\Delta t} \rfloor - 1}^{\lfloor \frac{t^\star_{q,k}}{\Delta t} \rfloor + 2} \eta_{\nu,\beta} B_\nu(s^\star_{q,k}) B_\beta(t^\star_{q,k}) \ .
\end{aligned}$$

where Δt and Δs are the grid sizes in the reference space. Thus, the ρ vector can be expressed as the result of a matrix-vector multiplication $\rho = M\,\eta$ where η represents the values of the distribution function. Furthermore, the matrix M depends only on the $B_\nu(s^\star_{q,k})B_\beta(t^\star_{q,k})$ products that are available at the beginning of the simulation and remain fixed over time. So, we precompute the matrix M initially and perform sparse matrix-vector multiplication each time we need the density ρ. Let us remark that the size of M only depends on the number N_v of samples along the v direction. To improve the accuracy of the integrals, one simply needs to perform a bigger precomputation to get the matrix M.

4. Results

4.1. **Geometry settings and experimental results**

We consider the evolution of a non stationary beam with the following dimensionless parameters:

$$N_0 = 2\pi, a = 1.83271471003, v_{th} = 0.0727518214392, K_{self} = 1., K_{app} = 0.5 \ .$$

In our simulations, the beam is transported over 163 lattice periods, with 100 time steps per period. The underlying uniform 2D grid used for the simulation consists in 2^{18} points covering entirely or partially the spatial domain $(r, v_r) \in [-6, 6] \times [-2.5, 2.5]$. We intend to compare the speed and accuracy on three geometries:

(1) *Original LOSS*: no diffeomorphism is employed and this case corresponds to the initial simulator configuration. The computational grid is rectangular and its size is 512×512.
(2) *Analytical mapping*: an analytical shaping function (see section 2.2) maps an oval in phase space to a reference domain $[0, 1] \times [0, 1]$. The reference domain has a uniform grid mesh of size 1024×256 with a larger number of points in the angle direction. Especially in this configuration, the periodicity along the angle direction has to be taken into account.
(3) *Bezier mapping*: a set of Bézier patches (see section 2.3) are built; the shape of the computational domain is in-between the shapes of the previous oval and rectangle configurations. The reference domain has a 512×512 size.

The different computational grids are sketched in figure 5 where an undersampling has been applied in order to improve readability.

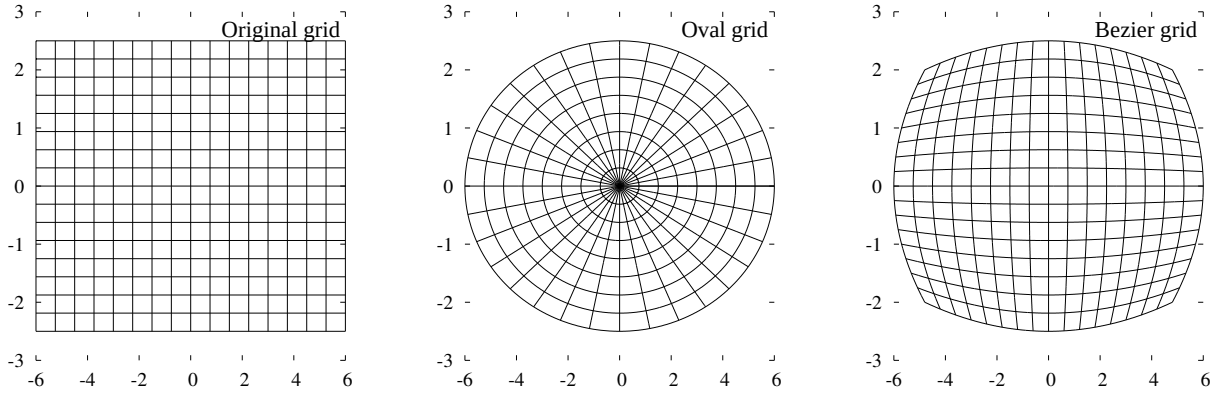
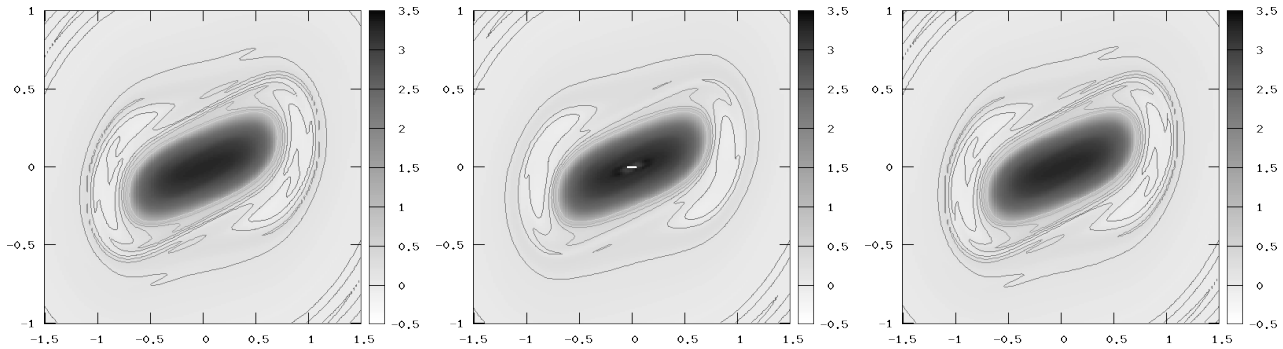


FIGURE 5. Three computational grids are tested for the paraxial beam problem: (a) regular cartesian grid, (b) oval grid associated to an analytical inverse mapping, (c) grid defined by Bézier elements



(a) Original (b) Analytic/Oval (c) Bézier

FIGURE 6. Final state of the distribution function in phase space in the three configurations

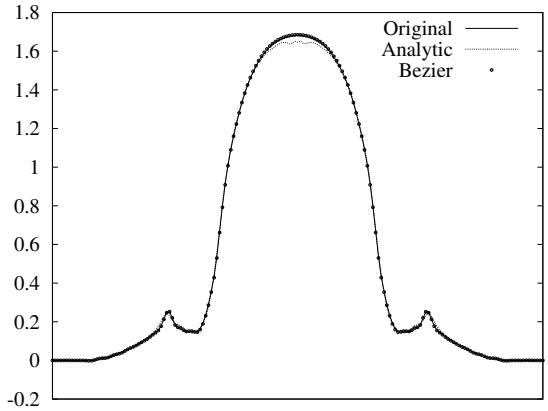


FIGURE 7. Final state of the density function $\rho(r)$ in the three configurations

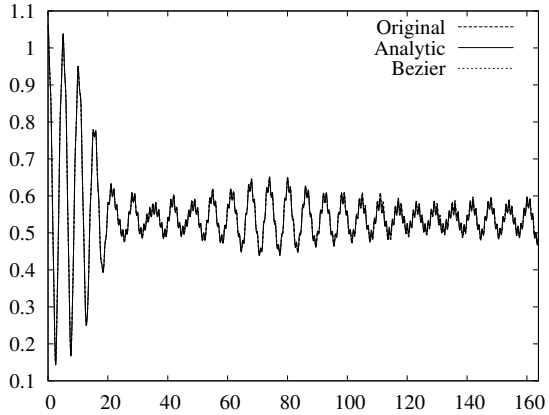


FIGURE 8. Evolution in time of the R_{rms} quantity in the three geometry configurations

The final state of the distribution function is shown in figure 6 for the different computational grids. For each configuration, the simulator manages to follow the evolution of filamentation. The distributions functions are quite similar, but the analytical mapping exhibits some clear differences with the two other settings. The strong anisotropy of the oval mapping in the center of the domain clearly triggers numerical problems.

These problems are also illustrated in figure 7 where the final state of density $\rho(r)$ is presented in the three geometries. The Bézier and the original curves almost overlay, but the curve for the analytical case has a different behavior at the peak of the density and on the lateral small shoulders. In figure 8, we consider the root mean square fluctuations of the spatial coordinate, i.e. $R_{rms} = (\int \rho(r) r^2 dr / \int \rho(r) dr)^{1/2}$, which measures the effective beam focalization. The three configurations give nearly identical R_{rms} curves over time. Finally, we conclude that diffeomorphims do not decrease accuracy directly, but care must be taken in order to avoid unjustified anisotropy.

4.2. **Performance issues**

Performance of the different geometry settings has been investigated on a desktop computer. A 3.0 GHz Intel Core 2 Duo E8400 processor was used to run numerical experiments on only one core. The settings of the runs were exactly the same as in the previous subsection. In Table 1, the timings of each part of the code for one simulation of 16384 time steps are gathered. The `Field solve` column sums time used to compute the density ρ and the self-consistent field. The `Spline coeff.` column corresponds to calls to the LU solver described in section 1.2. The `Advection` column comprises the trajectories computation to find the origins of the characteristics and the interpolation costs. The `Total` column sums the first columns, ignoring the preprocessing phase of the simulation and the diagnostics.

	Field solve	Spline coeff.	Advection	Total
Original LOSS	2.3	16.4	78	97
Analytical mapping	10.1	18.4	115	143
Bezier mapping	9.2	17.3	275	301

TABLE 1. Timing results (seconds) for three geometry settings and simulation over 1024 time steps

The field solver and ρ calculation take much more time when a mapping is employed. This is due to the fact that without mapping the computation $\rho(r)$ is very light, as it is only a sum of $f(r, v)$ along dimension v; whereas with mapping the strategy explained in subsection 3.3 is used. The computation of spline coefficients takes nearly the same amount of time in each case. The computation costs are almost identical and cache effects are the critical factor that modulates slightly the costs of computing spline coefficients. Concerning the advection step, the cost of the inverse mapping explains the observed differences in computation time. The analytical inverse mapping involves a 50% time increase compared to the original LOSS, and Bézier inverse mapping leads a 250% time increase.

If we look forward to the application of this technique in the GYSELA code, this overhead must be rescaled. The 2D interpolations represents approximately 5% of the execution time in a GYSELA run. A rough estimate of the overhead induced by implementing the Bézier setting in the GYSELA Vlasov solver is thus 13% (with the simple calculation $[275/78 - 1] \times 5\%$). Other overheads should also be considered for the field solver using the upgraded geometry setting, but no simple estimate of this cost can be provided for the moment.

CONCLUSION

The Semi-Lagrangian scheme combined with isoparametric analysis successfully solves the Vlasov equation on a reduced beam test case. Small-scale filamentations in phase-space are well captured by the new solver. Quantitative analysis shows that only minor issues in accuracy are caused by the mesh geometry. For a parametric Bézier patch, the overhead in terms of computational cost is about 200% on the whole simulation run when compared to the original version. Although this figure may appear quite large, it could in fact be reasonable if the advantages brought by a more accurate description of the domain geometry outweigh its

numerical cost. This should be the case for applications with strong geometrical constraints on the computational mesh, such as the gyrokinetic code GYSELA.

References

[1] P. Angelino, X. Garbet, and *al.* Role of Plasma Elongation on Turbulent Transport in Magnetically Confined Plasmas. *Physical Review Letters*, 102(19), 2009.

[2] A. Back, A. Crestetto, A. Ratnani, and E. Sonnendrücker. ISOPIC: coupling an axysimmetric PIC solver with the Isogeometric approach. In *In Proceedings of ESAIM*, 2010. In preparation.

[3] N. Besse and M. Mehrenberger. Convergence of classes of high-order semi-Lagrangian schemes for the Vlasov-Poisson system. *Mathematics of Computation*, 77(61):93–123, 2008.

[4] C.K. Birdsall and A. Langdon. *Plasma physics via computer simulation.* McGraw-Hill, New York, NY, USA, 1985.

[5] P. G. Ciarlet and P. A. Raviart. Interpolation theory over curved elements, with applications to finite element methods. *CMAME*, 1:217–249, 1972.

[6] N. Crouseilles, G. Latu, and E. Sonnendrücker. Hermite spline interpolation on patches for parallelly solving the Vlasov-Poisson equation. *Int. J. of Applied Math. and Computer Science*, 17(3):335–349, 2007.

[7] N. Crouseilles, G. Latu, and E. Sonnendrücker. A parallel Vlasov solver based on local cubic spline interpolation on patches. *Journal of Computational Physics*, 228(5):1429–1446, 2009.

[8] C. DeBoor. *A practical guide to splines.* Springer-Verlag, New York, applied mathematical sciences 27 edition, 2001.

[9] G. Farin. *Curves and surfaces for CAGD: a practical guide.* Morgan Kaufmann Pub. Inc., San Francisco, CA, USA, 2002.

[10] F. Filbet and E. Sonnendrücker. Modeling and numerical simulation of space charge dominated beams in the paraxial approximation. *Mathematical Models and Methods in Applied Sciences*, 16(5):763, 2006.

[11] F. Filbet, E. Sonnendrücker, and P. Bertrand. Conservative Numerical Schemes for the Vlasov Equation. *Journal of Computational Physics*, 172(1):166 – 187, 2001.

[12] X. Garbet, Y. Idomura, L. Villard, and T.-H. Watanabe. Gyrokinetic simulations of turbulent transport. *Nuclear Fusion*, 50(4):043002, 2010.

[13] M. Geimer and O. Abert. Interactive ray tracing of trimmed bicubic bézier surfaces without triangulation. In *Proceedings of WSCG*, pages 71–78, 2005.

[14] A.H. Glasser, I.A. Kitaeva, V.D. Liseikin, V.S. Lukin, and A.N. Simakov. Harmonic grid generation for the tokamak edge region. In *Proceedings of EPS, Spain*, 2005.

[15] V. Grandgirard, M. Brunetti, P. Bertrand, N. Besse, X. Garbet, P. Ghendrih, G. Manfredi, Y. Sarazin, O. Sauter, E. Sonnendrücker, J. Vaclavik, and L. Villard. A drift-kinetic Semi-Lagrangian 4D code for ion turbulence simulation. *Journal of Computational Physics*, 217(2):395–423, 2006.

[16] V. Grandgirard, Y. Sarazin, P. Angelino, A. Bottino, N. Crouseilles, G. Darmet, G. Dif-Pradalier, X. Garbet, P. Ghendrih, S. Jolliet, et al. Global full-f gyrokinetic simulations of plasma turbulence. *Plasma Phys. and Control. Fusion*, 49:B173, 2007.

[17] T.J.R. Hughes, J.A. Cottrell, and Y. Bazilevs. Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, 194(39-41):4135 – 4195, 2005.

[18] B.M. Irons. Engineering application of numerical integration in stiffness method. *Journal of the American Institute of Aeronautics and Astronautics*, 4:2035–2037, 1966.

[19] S. Jolliet, A. Bottino, P. Angelino, R. Hatzky, TM Tran, BF McMillan, O. Sauter, K. Appert, Y. Idomura, and L. Villard. A global collisionless PIC code in magnetic coordinates. *Computer Physics Communications*, 177(5):409–425, 2007.

[20] M Lenoir. Optimal isoparametric finite elements and error estimates for domains involving curved boundaries. *SIAM J. Numer. Anal.*, 23:562–580, June 1986.

[21] D. Lischinski and J. Gonczarowski. Improved techniques for ray tracing parametric surfaces. *The Visual Computer*, 6, 1990.

[22] T. Nishita, T. W. Sederberg, and M. Kakimoto. Ray tracing trimmed rational surface patches. *SIGGRAPH Comput. Graph.*, 24(4):337–345, 1990.

[23] B. Smits. Efficiency issues for ray tracing. *J. Graph. Tools*, 3:1–14, February 1998.

[24] E. Sonnendrücker, F. Filbet, A. Friedman, E. Oudet, and J.-L. Vay. Vlasov Simulations of beams with a moving grid. *Computer Physics Communications*, 164(1–3):390–395, 2004.

[25] E. Sonnendrücker, M. Gutnic, M. Haefele, G. Latu, and J.L. Lemaire. Vlasov Simulation of Beams and HALO. In *Proceedings of the Particle Accelerator Conference, 2005*, pages 581–585, 2005.

[26] E. Sonnendrücker, J. Roche, P. Bertrand, and A. Ghizzo. The Semi-Lagrangian Method for the Numerical Resolution of the Vlasov Equation. *Journal of Computational Physics*, 149(2):201–220, 1999.

[27] A. Staniforth and J. Côté. Semi-lagrangian integration schemes for atmospheric models: A review. *Monthly Weather Review*, 119(9):2206–2223, 1991.

[28] I.C. Taig. Structural analysis by the matrix displacement method. *English Electric Aviation Report*, SO 17, 1961.

[29] D. L. Toth. On ray tracing parametric surfaces. *SIGGRAPH Comput. Graph.*, 19(3):171–179, 1985.

[30] S.W. Wang, Z.C. Shih, and R.C. Chang. An efficient and stable ray tracing algorithm for parametric surfaces. *J. Inf. Sci. Eng.*, 18(4):541–561, 2002.