Lagrangian Representations of Flow Fields with Parameter Curves

Roxana Bujack*
University of California Davis

Kenneth I. Joy†
University of California Davis

ABSTRACT

There are two main principles in which a flow field can be described. In its Eulerian representation, the immediate direction of the flow is used. In its Lagrangian representation, the paths of fluid parcels are stored. The latter way has proven less prone to error in cases when only a sparse subset of the original flow field can be used due to restrictions in computational capacity, which are inevitable in practice. But so far, a main drawback of the Lagrangian representation is that it leads to unsightly lines with sharp edges, because the spanning trajectories are stored as polygonal chains to minimize storage space.

In this paper, we tackle this issue by proposing to represent the trajectories by means of parameter curves, like composite Bézier curves and cubic Hermite splines, instead of polygonal chains. We demonstrate that a parameter representation results in smoother lines with comparable approximation error under equal conditions of limited storage capacity.

1 INTRODUCTION

In fluid dynamics, flow is a set of continuous, time-dependent physical quantities in a spatial domain (density, pressure, stress, temperature, and flow velocity) that describe the behavior of a fluid in motion. Among these, the velocity is most commonly used for visualization. There are two specifications of a flow field. On one hand, the Eulerian specification describes the flow passing through a fixed spatial domain. It can be interpreted as the point of view of a stationary observer standing at a river's bank watching the water flow by. The usual way of storing a flow field in its Eulerian specification is by means of its velocity field, which is a time-dependent vector field that maps each point in space to the velocity of the flow at a given time

$$v : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d, \quad x,t \mapsto v(x,t). \tag{1}$$

On the other hand, the Lagrangian specification of a flow field describes the properties of a fixed fluid parcel as it travels through space. This specification can be imagined as the point of view of an observer that sits in a boat and moves along with the flow of the river. The usual way of storing a flow field in its Lagrangian specification is by means of its flow map $F_{t_0}^t$. A flow map identifies starting positions x_0 at times t_0 of massless particles with their trajectories as they are advected by the flow, which means that the particle moves tangentially to the flow velocity at all times. Mathematically, the flow map is the mapping

$$F_{t_0}^t : \mathbb{R} \times \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}^d, \quad t \times t_0 \times x_0 \mapsto F_{t_0}^t(x_0) = x(t) \tag{2}$$

of initial values x_0 to the solutions of the ordinary differential equation

$$\frac{d}{dt}x(t) = v(x(t),t). \tag{3}$$

*e-mail: bujack@ucdavis.edu
†e-mail: joy@cs.ucdavis.edu

Practically, a flow parcel at (x_0,t_0) moves to $F_{t_0}^t(x_0)$ in the time interval $[t_0,t]$. Once the trajectories are known, storing the velocity is no longer necessary because it is indirectly given through the latter relation.

Theoretically, the two representations are equivalent because the trajectories can be derived from the vectors through integration and the vectors from the trajectories through differentiation. In practice however, there is a big difference. One point to consider is that either representation will be given at discrete positions and times only. Evaluations of the flow at arbitrary positions have to be estimated using interpolation. The other issue is that the integration or differentiation to convert one representation to the other have to be numerically approximated.

The flow fields are usually simulated on high performance supercomputers with many small time steps to keep error accumulation, which appears in numerical integration, small. In modern supercomputer architecture the computational capacity is much bigger than the I/O bandwidth [7]. That is why only few time steps of the calculation are actually flushed out and stored to disk. In order to get as much information as possible through this bottleneck, the Lagrangian representation has proven more useful, especially under extreme temporal sparsity [1]. The reason is that a majority of flow visualization techniques use trajectories and that the numerical integration in settings of temporal sparsity is prone to error propagation.

There is one major drawback though in the way the trajectories have been handled so far. It is their lack of smoothness. In order to minimize the storage requirements, the trajectories have been stored just through their endpoint in the consecutively stored time step. For the generation of arbitrarily seeded, longer pathlines, the interpolated trajectories are patched after every time step. The result is a non-smooth, unsightly polygonal chain. In this paper, we analyze how parametric curves can be used to overcome this issue.

2 RELATED WORK

Advection has been used for flow visualization much longer than computers [30]. Since the seminal work of Hulquist [16], it has been the primary technique for calculating the trajectory of a particle in a flow field. In state-of-the-art flow visualization, the large majority of techniques utilize advection [19, 20, 27, 21, 24, 3]. Advection methods typically construct integral curves, which are continuous functions tangential to the vector field. An integral curve encodes the trajectory of a single massless particle, which in turn gives insight into the flow behavior in the area surrounding the particle's path. When considering many integral curves throughout the spatial domain, the flow field starts to be revealed, and is ultimately defined in terms of a flow map. When the field is defined by a flow map, then the representation is referred to as Lagrangian.

Lagrangian methods have been utilized significantly within the flow visualization community over the last few years through the calculation of Lagrangian Coherent Structures (LCS) by Haller et al. [14, 13], which focus on features of the flow field based on the calculation of stable manifolds. These time-consuming methods, largely-limited by the number of advection steps required, have been improved by the visualization community through GPU acceleration [11, 25], adaptive mesh refinement techniques [10], and interpolation over sparse particles [2]. Others have used the La-

grangian methods by incorporating flow maps into the Eulerian representation of a flow field [28, 26, 17].

Hlawatsch et al. [15] and Agranovsky et al. [1] use the flow map directly. Hlawatsch et al. utilize a hierarchical scheme to decrease the number of integration steps by constructing longer integral lines from previously computed partial solutions. They precompute a set of Lagrangian-based trajectories and attempt to choose the correct ones to utilize when forming pathlines. Agranovsky et al. utilize trajectories at each point of a mesh, interpolating these trajectories by utilizing barycentric coordinate interpolation over the mesh elements. They argue that their method is faster, more accurate, and uses less memory to obtain a similar error to advection methods in the context of in-situ calculation, where all the results of a simulation are known and trajectories can be explicitly analyzed. In addition, by using Lagrangian representations, this allows their method to avoid problems in time sparsity.

Chandler et al. [5] construct visualizations of particle based flows and utilize trajectories of the particles in their calculations. They limit their study to smoothed particle hydrodynamics (SPH) applications [22], where each particle is associated with a parameter that defines a radius about which they influence the flow. Chen et al. use Bézier curves for the visualization of uncertainty [6].

The method of Agranovsky et al., in its simplest form, saves the endpoint of each trajectory for each mesh element – thereby not increasing the storage for the method. Using only these endpoints, pathlines generated through this method must be piecewise linear, violating the smooth integral curve visualization that we anticipate in the visualization field. This paper presents an analysis of representing these trajectories not as single points, but as curves – interpolating these curved trajectories to obtain the pathlines necessary for integration into visualization methods. We explore various methods for generation of these curves and analyze their corresponding errors.

3 Parameter Representations of Trajectories

Mathematically, a trajectory is a function mapping a time interval onto a curve, which is a one-dimensional subset in space

$$\mathbb{R} \to \mathbb{R}^d, \quad t \mapsto x(t). \tag{4}$$

This function can be given by an implicit equation or a parameterization. In a discrete setting, the points can also be given explicitly.

We will work with parameter representations and especially concentrate on two of the most popular parametric curves: the composite cubic Bézier curves and the cubic Hermite splines. They are easy to handle, flexible but robust, and have proven very useful in computer graphics and computer aided geometric design. The main reason for us to choose them is that they consist of cubic polynomials. These are the lowest order polynomials that can be concatenated to form smooth curves in which changes have only local influence. They also are the lowest order polynomials that can have points of infliction and the lower the order, the higher the robustness the polynomial. For an introduction to parameter curves, we recommend [8]. Now, we give a brief overview on the ones that we will analyze.

3.1 Polygonal Chains

The probably easiest way to describe a trajectory is by means of a polygonal chain or piecewise linear curve. This is a number of connected line segments, defined by their endpoints $p_0, ..., p_n \in \mathbb{R}^d$ at given times $t_0, ..., t_n \in \mathbb{R}$. The trajectory $x(t) \subset \mathbb{R}^d$ is then constructed from linear interpolation of the preceding and following given points in time. That means for $t_i \leq t \leq t_{i+1}$, it suffices

$$x(t) = \frac{t-t_i}{t_{i+1}-t_i} p_i + (1 - \frac{t-t_i}{t_{i+1}-t_i}) p_{i+1}. \tag{5}$$

Especially, the points $p_0, ..., p_n$ are part of the trajectory. The resulting curve is continuous, but in general not differentiable, i.e. it is a C^0 curve and not smooth. An example can be found in Figure 2(a).

3.2 Cubic Hermite Splines

A piecewise linear curve can be transformed into a smooth curve using a cubic Hermite spline [8, 4, 18]. It passes through all the points in the sequence, but instead of combining the points linearly, it uses cubic polynomials in each segment. The additional degrees of freedom are used to tune the derivatives on both sides of each point.

If a general cubic polynomial is given in its Hermite basis, the coefficients immediately coincide with the given points $q_0, q_1 \in \mathbb{R}^d$ and derivatives $\dot{q}_0, \dot{q}_1 \in \mathbb{R}^d$ at both ends

$$\begin{aligned} x(t) =& q_0(2t^3 - 3t^2 + 1) + \dot{q}_0(t^3 - 2t^2 + t) \\ &+ q_1(-2t^3 + 3t^2) + \dot{q}_1(t^3 - t^2). \end{aligned} \tag{6}$$

Using the Hermite basis makes the construction of the whole spline for the sequence $p_0, ..., p_n$ very easy. The claim for the spline to pass through the interpolation points fixes two of four degrees of freedom in each segment. Since we do not have information about the derivatives, we have to reasonably estimate them from the data. This problem has no generally valid optimal solution and different choices lead to different splines. To guarantee differentiability in each point, we average the forward and backward finite differences at the ends of each spline segment. In particular, using four consecutive points $p_{i-2}, p_{i-1}, p_i, p_{i+1} \in \mathbb{R}^d$ with parameter values $t_{i-2}, t_{i-1}, t_i, t_{i+1} \in \mathbb{R}$, we can construct the i-th segment between p_{i-1} and p_i by setting

$$\begin{aligned} q_0 =& p_{i-1}, \\ q_1 =& p_i, \\ \dot{q}_0 =& \frac{1}{2}\left(\frac{p_i - p_{i-1}}{t_i - t_{i-1}} + \frac{p_{i-1} - p_{i-2}}{t_{i-1} - t_{i-2}}\right)(t_i - t_{i-1}), \\ \dot{q}_1 =& \frac{1}{2}\left(\frac{p_i - p_{i-1}}{t_i - t_{i-1}} + \frac{p_{i+1} - p_i}{t_{i+1} - t_i}\right)(t_i - t_{i-1}) \end{aligned} \tag{7}$$

in (6) to get an overall C^1 spline. The first and the last segment need to be treated differently because there is only one neighboring point. We just use the one-sided finite differences there. In the case of equidistant times, this C^1 spline coincides with the well known Catmull-Rom spline [4]. An example can be found in Figure 2(b).

3.3 Bézier Curves

One of the most popular parametric curve representations in computer graphics are Bézier curves [8]. They were independently developed by Pierre Bézier at Renault and Paul de Casteljau at Citroën in the early 1960s.

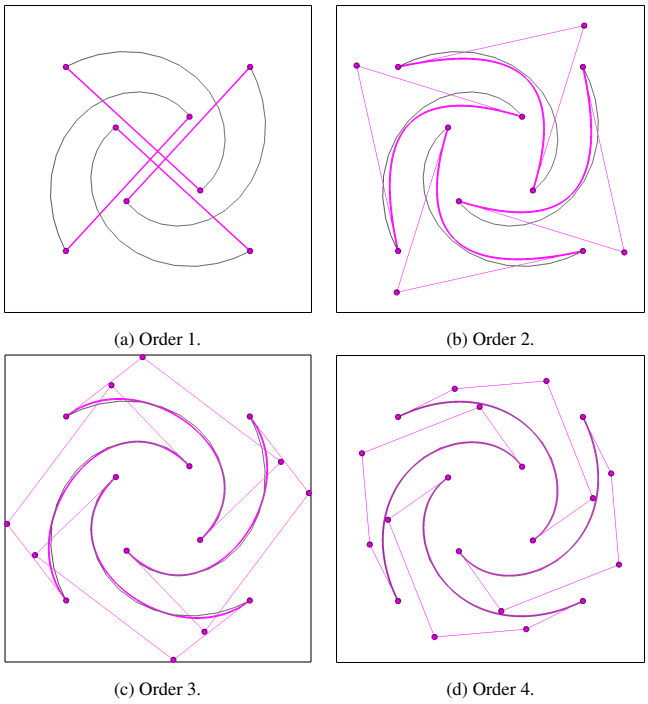
A Bézier curve of order $n \in \mathbb{N}$ over the parameter $t \in [0,1]$ is defined by a sequence of $n+1$ control points $p_0, ..., p_n \in \mathbb{R}^d$ and the calculation rule

$$x(t) = \sum_{i=0}^{n} B_{i,n}(t) p_i \tag{8}$$

with the Bernstein polynomial

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}. \tag{9}$$

The endpoints p_0, p_n lie on the Bézier curve, but the remaining control points generally not.



(a) Order 1. (b) Order 2.

(c) Order 3. (d) Order 4.

Figure 1: The pink curves are the best approximations of Bézier curves of order 1 to 4 to the original grey trajectories in the sense of a least squares fit. The thin pink lines connect the control points. The endpoints are not part of the least squares variation.

An illustration of the approximation capabilities of the lowest order Bézier curves can be found in Figure 1 applied to some example pathlines in a simple spiral flow field.

The most popular Bézier curves are the cubic ones. Defined by four points $p_0, \ldots, p_3$, they take the shape

$$x(t) = (1-t)^3 p_0 + 3t(1-t)^2 p_1 + 3t^2(1-t)p_2 + t^3 p_3. \quad (10)$$

Their behavior is very intuitive. The two outer control points define the endpoints of the curve while the two control points in the middle steer the derivatives of the curve at its ends, because of

$$\begin{aligned} x'(0) &= 3(p_1 - p_0), \\ x'(1) &= 3(p_3 - p_2). \end{aligned} \quad (11)$$

3.4 C^0 Composite Bézier Curves

In order to get a more flexible curve descriptor based on low order Bézier curves, composite Bézier curves are used [8]. Similar to the piecewise linear curves in Subsection 3.1, the final curve is constructed from a sequence of connected line segments. Only that in this case, we do not use piecewise linear curves but piecewise Bézier curves. The result is a continuous C^0 curve.

Because of their flexibility and simplicity, especially composite cubic Bézier curves are used to describe vector graphics in Corel Draw, Gimp, and Inkscape and also fonts, like for example, PostScript. A composite cubic Bézier curve is defined by a sequence of points $p_0, \ldots, p_{3n+1}$. For each $i \in \mathbb{N}$ the control points $p_{3i}, p_{3i+1}, p_{3i+2}, p_{3i+3} \in \mathbb{R}^d$ form one cubic Bézier curve. The curve is C^0. Every third point $p_{3i} \in \mathbb{R}^d$ in the sequence lies on the curve and is associated with a time $t_{3i} \in \mathbb{R}$. An example can be found in Figure 2(c).

3.5 C^1 Cubic composite Bézier Curves

The interpretation of the cubic Bézier curves to have the positions fixed by the outer control points and the derivatives fixed by the inner control points makes it easy to construct smooth composite Bézier curves from them.

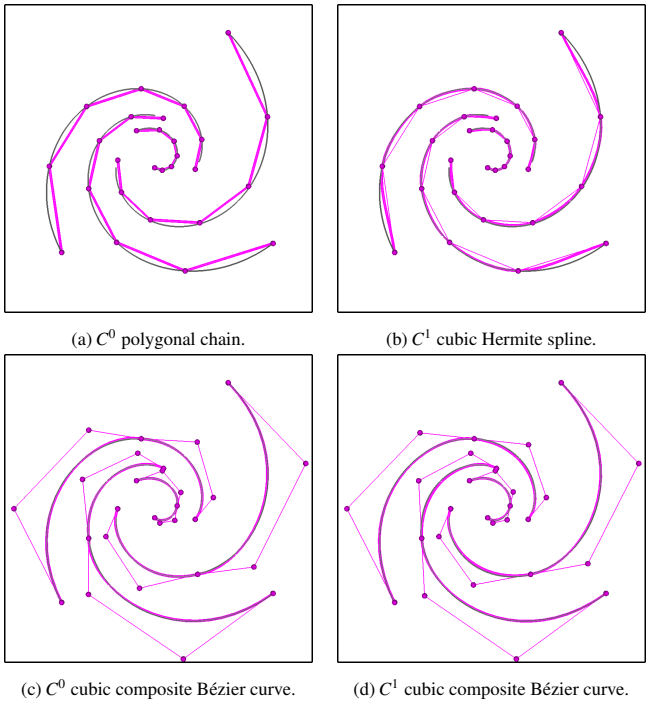
Similar to the C^1 spline, we can slightly modify the control points of two consecutive segments to construct a C^1 curve. Let p_{3i} at parameter t_i be the endpoint of the i-th segment and the start-point of segment $i+1$, then we average the left and right derivative

$$\begin{aligned} \dot{x}_-(t_i) &= 3\frac{p_{3i} - p_{3i-1}}{t_{3i} - t_{3i-3}}, \\ \dot{x}_+(t_i) &= 3\frac{p_{3i+1} - p_{3i}}{t_{3i+3} - t_{3i}}, \\ \dot{x}(t_i) &= \frac{1}{2}(\dot{x}_-(t_i) + \dot{x}_+(t_i)) \end{aligned} \quad (12)$$

and adjust the positions of the neighboring control points respectively

$$\begin{aligned} p'_{3i-1} &= p_{3i} - \frac{1}{3}\dot{x}(t_i)(t_{3i} - t_{3i-3}), \\ p'_{3i+1} &= p_{3i} + \frac{1}{3}\dot{x}(t_i)(t_{3i+1} - t_{3i}). \end{aligned} \quad (13)$$

An example can be found in Figure 2(d).



(a) C^0 polygonal chain. (b) C^1 cubic Hermite spline.

(c) C^0 cubic composite Bézier curve. (d) C^1 cubic composite Bézier curve.

Figure 2: Overview of the used trajectory representations on a spiral flow example. The grey pathline is approximated by the thick pink curve. The thin pink lines connect the control points.

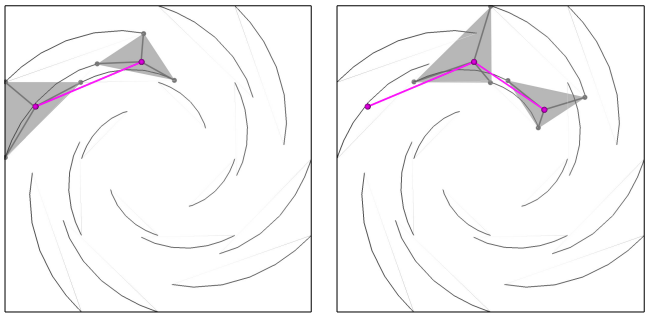
4 Work Flow

Similarly to the work of Agranovsky et al. [1] , the following steps have to be made to generate arbitrary pathlines from the trajectory representations.

1. Calculation of the spanning trajectories.
2. Calculation of the parameters for storing the spanning trajectories depending on the different curve representations.

3. Construction of randomly seeded pathlines from interpolation and composition of the spanning trajectories.
4. Evaluation of the pathlines.

If the numerical flow simulation works with trajectories, the first step is already done with the high temporal resolution of the simulation using N_t^{sim} time steps. If not, it has to be done simultaneously with the simulation, but the additional computational effort is small compared to the costs of the simulation. The second step is also to be performed in-situ. Then, we face the bottleneck when the parameters of the curves are stored for only $N_t < N_t^{sim}$ time slices. The two latter steps are done post-hoc, independently from the simulation. They work on the sparser information that could be stored.



(a) First time step. (b) Second time step.

Figure 3: Spanning trajectories are drawn in dark grey. They are approximated by lines, which are shown in light grey. The randomly seeded startpoint (pink) is interpolated using the closest spanning trajectories moving from the left grey triangle to the the right grey triangle. This step is repeated until the desired length for the pathline is reached.

In our experiments, we work with analytic flow fields and perform all four steps. We use the Runge Kutta integration method on the analytically defined vector fields to generate the spanning trajectories. We restart them seeded on a regular grid after every of the N_t I/O operations to prevent areas from running empty. Since the spanning trajectories are short compared to the whole integration time, we store them es elementary curves. For the splines, we store them as line segments. That means we store their endpoints. For the composite Bézier curves, we determine the optimal Bézier curve from solving the least squares fit and store their control points. When we construct the randomly seeded pathlines, we will linearly interpolate these short trajectories for the interval of each time slice N_t. That means, we interpolate the endpoints and control points respectively. Then, we compose the result to produce long pathlines as can be seen in Figure 3. On a uniform grid, bi-linear or tri-linear interpolation can be used and barycentric coordinates for irregularly distributed spanning trajectories, as they occur in SPH simulations.

5 Theory

In this section, we analyze the theoretical bounds on the error using numerical mathematics.

5.1 Tools

We will use the well known Taylor series for $f : \mathbb{R}^d \to \mathbb{R}$

$$f(x) = \sum_{|\alpha|<k} \frac{D^\alpha f(a)}{\alpha!}(x-a)^\alpha + \sum_{|\alpha|=k} \frac{D^\alpha f(\zeta)}{\alpha!}(x-a)^\alpha, \quad (14)$$

with the multi-index $\alpha \in \mathbb{N}^d$ and $\zeta \in B_{\|x-a\|}(x)$, [9]. From the remainder of the Taylor series follows the estimate for the linear interpolation L_x of a function $f : \mathbb{R}^d \to \mathbb{R}$ with respect to x

$$|L_x f(x) - f(x)| \leq \frac{d^2}{8} h_x^2 \max_{\zeta \in B_{\|x-a\|}(x)} \|H_f(\zeta)\|_\infty. \quad (15)$$

with the Hessian matrix H_f containing all second order partial derivatives of f and the vector norm $\|.\|_\infty$ returning its maximal component, [29]. For a multi-dimensional function $f : \mathbb{R}^d \to \mathbb{R}^d$, the previous estimate holds for each of its components. Finally, we will use the following properties of the flow map

$$L_x F_{t_{j-1}}^{t_{j-1}}(x) = x = F_{t_{j-1}}^{t_{j-1}}(x) \quad (16)$$

and because of that, the second derivatives vanish

$$H_{F_{t_{j-1}}^{t_{j-1}}}(x) = 0. \quad (17)$$

5.2 Error at the Stored Times

This estimate holds for all discussed parameter representations of the trajectories, because they have the same endpoints at the stored time steps. Using the Taylor expansion with respect to time, we can write

$$L_x F_{t_{j-1}}^{t_j}(x) \overset{(14)}{=} L_x F_{t_{j-1}}^{t_{j-1}}(x) + h_t L_x \dot{F}_{t_{j-1}}^{\tau}(x), \quad (18)$$

which reveals that patching together the interpolated trajectories is a special kind of one-step numerical integration method

$$x_j = x_{j-1} + h_t f(t_{j-1}, x_{j-1}) \quad (19)$$

with increment function

$$f(t_{j-1}, x_{j-1}) = L_x \dot{F}_{t_{j-1}}^{\tau}(x_{j-1}). \quad (20)$$

Its local approximation error τ_n suffices

$$\begin{aligned} \|L_x F_{t_{j-1}}^{t_j}(x) - F_{t_{j-1}}^{t_j}(x)\|_\infty &\overset{(15)}{\leq} \frac{d^2}{8} h_x^2 \|H_{F_{t_{j-1}}^{t_j}}(\zeta)\|_\infty \\ &\overset{(14)}{=} \frac{d^2}{8} h_x^2 \|H_{F_{t_{j-1}}^{t_{j-1}}}(\zeta) + h_t H_{\dot{F}_{t_{j-1}}^{\tau}}(\zeta)\|_\infty \\ &\overset{(17)}{=} \frac{d^2}{8} h_x^2 h_t \|H_{\dot{F}_{t_{j-1}}^{\tau}}(\zeta)\|_\infty. \end{aligned} \quad (21)$$

It is known from numerics of ordinary differential equations [12] that the global truncation error

$$e_n = x_n - x(t_n) \quad (22)$$

is related to the local truncation error by

$$e_n \leq n \max_{j=0,\ldots,n} \tau_j e^{L(t_n - t_0)} \quad (23)$$

if the increment function is Lipschitz continuous with respect to its second argument with Lipschitz constant L. Our increment function is piecewise linear and therefore Lipschitz continuous. Together with $h_t n = t_n - t_0$, that leads to

$$e_n \leq \frac{d^2}{8}(t_n - t_0) h_x^2 \max_{\tau \in [t_0, t_n]} \max_{\zeta \in \mathbb{R}^d} \|H_{\dot{F}_{t_{j-1}}^{\tau}}(\zeta)\|_\infty e^{L(t_n - t_0)}. \quad (24)$$

This estimate is in $O(h_x^2)$ if the flow map has bounded second derivatives in space and first derivatives in time and shows no dependence on h_t. The resolution with respect to time does theoretically play a role only when it comes to the interpolation of the values between the stored time slices.

5.3 Error at Arbitrary Times

Between the stored time steps, each parameter representation has its own approximation error additionally to the global truncation error (24).

In case of the polygonal chain, we interpolate linearly L_t between the one-step points x_j with respect to time. The additional approximation error for arbitrary times $t \in [t_{j-1}, t_j]$ is given by

$$\|L_t F^t_{t_{j-1}}(x) - F^t_{t_{j-1}}(x)\|_\infty \leq \frac{1}{8} h_t^2 \max_{\tau \in [t_{j-1}, t_j]} \|\ddot{F}^\tau_{t_{j-1}}(x)\|. \tag{25}$$

Together with (24), that leads to an overall error in $O(h_t^2 + h_x^2)$ if all derivatives are bounded.

If we use the cubic Bézier curves from the least squares fit, we know that it performs at least as good as cubic interpolation C_t, which is known from [29] to suffice

$$\|L_t F^t_{t_{j-1}}(x) - F^t_{t_{j-1}}(x)\|_\infty \leq \frac{1}{24} h_t^4 \max_{\tau \in [t_{j-1}, t_j]} \|\dddot{F}^\tau_{t_{j-1}}(x)\|. \tag{26}$$

Together with (24), that leads to an overall approximation error of $O(h_t^4 + h_x^2)$ for arbitrary times.

As a result, the Bézier curves perform theoretically better than the piecewise linear curves for small time steps h_t. But it is not trivial to predict which variant is to be favored in practical applications because we have to deal with large h_t and the error of the smoothing displacement from C^0 to C^1 is not easily estimated. To decide which trajectory representation is best, experiments are necessary.

6 Experiments

In this section, we want to compare the smoothness and the approximation error of the interpolated pathlines depending on the different curve representations: the polygonal chain, the cubic Hermite spline, and the composite Bézier curves in their C^0 and C^1 versions.

6.1 Storage Space

For a given number of spanning trajectories in space N_x, the Bézier curves have to cover a time interval h_t^B of three times the length as the splines $h_t^B = 3h_t^S$. That way, they all use the same amount of storage space at the I/O bottle neck. Please note that if there is no regular grid, which occurs in smooth particle hydrodynamics (SPH) simulations, for example, the starting point has to be stored, too. In that case, the cubic Bézier curves need to cover only twice the temporal distance of the linear trajectories. In that case they would perform a little better than in our experiments on the regular grid.

6.2 Data

For the evaluation of the approximation error and the smoothness of the trajectory representations, we use four time-dependent analytic vector fields. The first two are highly artificial and have specially been chosen to challenge the higher order, smooth trajectory representations. The latter are common test problems that feature a more natural flow behavior.

Distorted Laminar Flow. Occasionally higher order approximation schemes tend to suffer from overfitting. To evaluate the sensitivity of the different approaches, we use a laminar flow that is distorted by random noise having 1% the intensity of the flow itself. An illustration can be found in Figure 5(a).

Turbulent Flow. Another potential shortcoming of smooth curve representations is that they are unable to approximate pathlines that are not smooth in the first place, as they appear in turbulent flow. To analyze the over smoothing, we use a random flow field, an impression of which can be found in Figure 5(b).

Double Gyre. The double gyre flow field is a two-dimensional dataset describing two counter-oriented vortices with periodically shifting vortex cores over time. It is given by the formula

$$v(x,y,t) = \begin{pmatrix} -A\pi \sin(\pi(f(x,t))\cos(\pi y) \\ A\pi \cos(\pi(f(x,t))\sin(\pi y)\frac{df}{dx} \end{pmatrix} \tag{27}$$

with

$$f(x,t) = \varepsilon \sin(2\pi\omega t)x^2 + (1 - 2\varepsilon \sin(2\pi\omega t)x. \tag{28}$$

In our experiments, we use the parameters $A = 0.3$, which limits the velocity of the field, $\varepsilon = 0.25$, which influences how far the vortices move to the left and the right, and $\omega = \frac{1}{10}$, which is the temporal frequency of the moving cores. The flow field is periodic in space, too. The pathlines seeded in $[0,2] \times [0,1]$ do not leave this area. We equidistantly seed the spanning trajectories on this domain and use a total simulation duration of $T = 10$, which coincides with one full periodic cycle in time.

ABC. The Arnold-Beltrami-Childress (ABC) dataset is a three-dimensional time-dependent flow field. It is a solution of Euler's equations in three dimensions for incompressible, inviscid fluid flows. Given by the analytic formula

$$v(x,y,z,t) = \begin{pmatrix} ((A + \frac{1}{2}t\sin(\pi t))\sin(z) + C\cos(y)) \\ (B\sin(x) + (A + \frac{1}{2}t\sin(\pi t))\cos(z)) \\ (C\sin(y) + B\cos(x)) \end{pmatrix}, \tag{29}$$

it is 2π-periodic along each coordinate axis and has a period of 2 with respect to time. We use the parameters $A = \sqrt{3}, B = \sqrt{2}, C = 1$, the domain $[0, 2\pi]^3$ and a total simulation duration of $T = 4$.

In contrast to the double gyre dataset, most of the pathlines actually leave the domain during the simulation. We stop the interpolation if the next step would lead outside the domain.

6.3 Approximation Error

To evaluate the approximation error of an interpolated pathline $x(t)$, we compare it to a numerically integrated pathline $\tilde{x}(t)$ starting at the same seedpoint using the same Runge Kutta integrator as for the calculation of the spanning trajectories on the analytic field. We measure the average squared Euclidean distance of the points on the interpolated and integrated pathlines at all N_t^{sim} time steps that were used for the integration

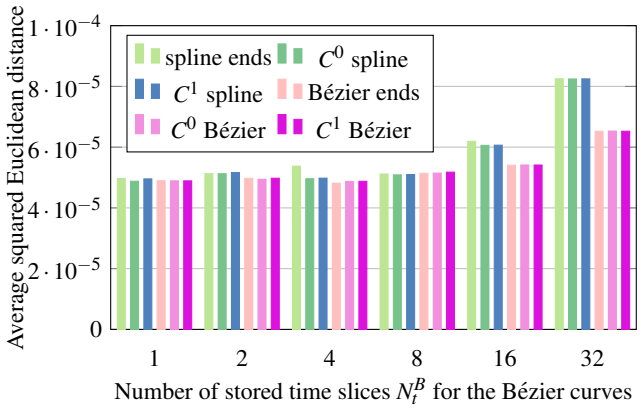
$$\frac{1}{N_t^{sim}} \sum_{i=0}^{N_t^{sim}} \|x(t_i) - \tilde{x}(t_i)\|_2^2. \tag{30}$$

In order to better distinguish how much of the error stems from the interpolation in space and how much from the different trajectory representations, we also calculate the average squared Euclidean distance only at the endpoints of each of the N_t stored time slices.

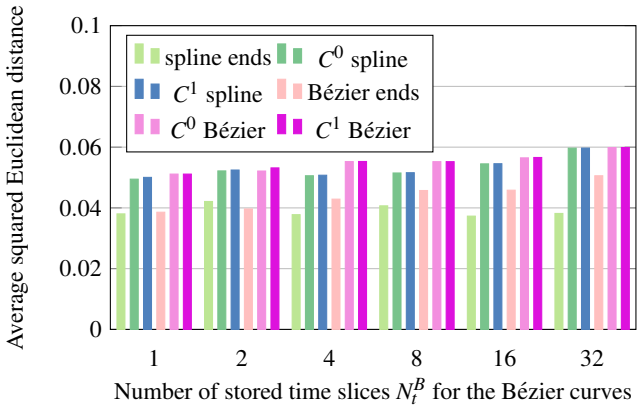
Figure 4 contains the approximation error of the four analytic datasets for fix but representative spacial resolutions. The x-axis displays the number of time slices that are stored to disk N_t^B for the Bézier curves. The splines always have $N_t^S = 3N_t^B$ stored time slices from 3 to 192.

The experiments with respect to approximation error show the following results:

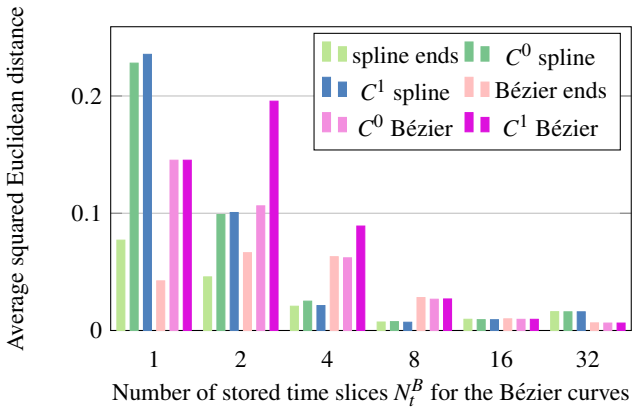
- The four curve representations give similar results for the laminar flow. The polynomials have still relatively low order are not very prone to overfitting.
- The results for the random flow differ hardly either. In the presence of turbulent behavior, the approximation quality at the given points is already very poor. Therefore, the over-smoothing does not significantly contribute to the error.



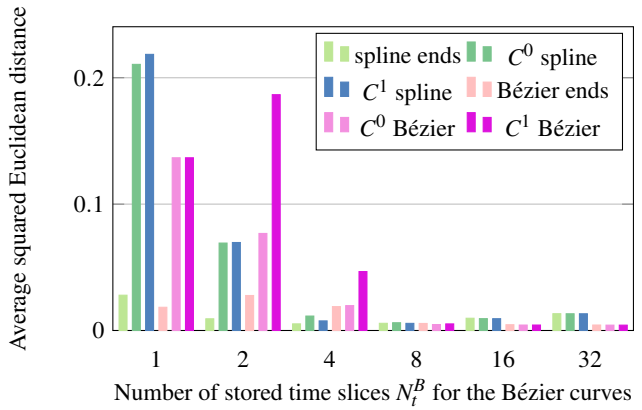
(a) Laminar flow with spacial resolution 50×50.



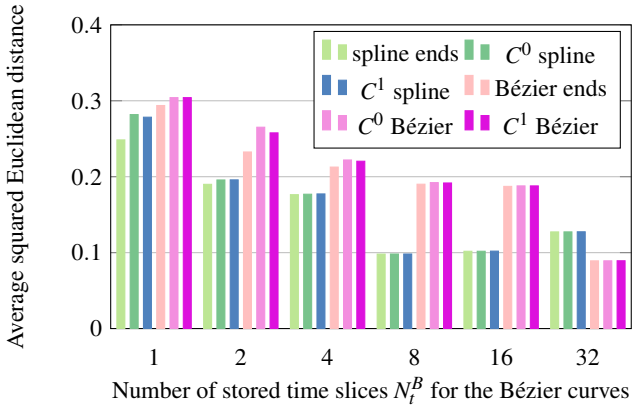
(b) Turbulent flow with spacial resolution 50×50.



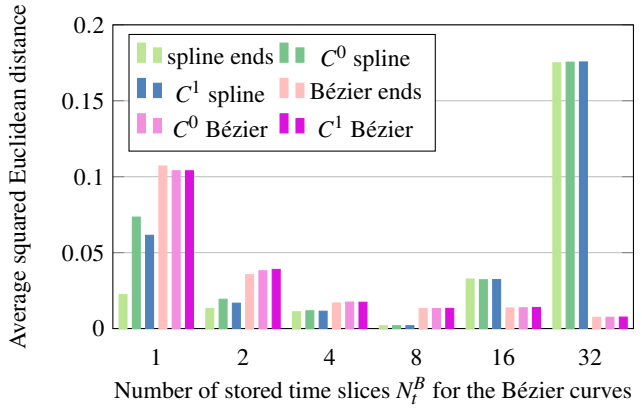
(c) Double gyre flow with low spacial resolution 50×25.



(d) Double gyre flow with higher spacial resolution 100×50.



(e) ABC flow with low spacial resolution $20 \times 20 \times 20$.



(f) ABC flow with higher spacial resolution $50 \times 50 \times 50$.

Figure 4: Experimantal results on the approximation error between the interpolated pathlines and the integrated ones depending on the number of stored time slices N_t^B for the different flow fields. The splines store three times the time slices $N_t^S = 3N_t^B$.

- The error drops with increasing storage capacity in the double gyre and ABC flow, but not in the laminar or turbulent flow.

- The error at the endpoints is higher for very small and very large values N_t. This observation seems to contradict the theoretical result, but can be explained as follows. There are small parts in the fields that have high divergence, where the interpolation suffices high errors. If we work with only one or two time steps, this will lead the trajectory to a very unreliable place. For lower step sizes on the other hand, the particle will move out of these regions quickly and not be disturbed as much. Especially particles that start close to the edges of the double gyre dataset have great errors because half of its neighbors never leave the edge. For very short time steps, the error also increases, which is due to the finiteness of the floating point representation and the fact that the spanning trajectories are not perfectly exact. That was not considered in the theoretical analysis.

- The approximation error of the different curve representations

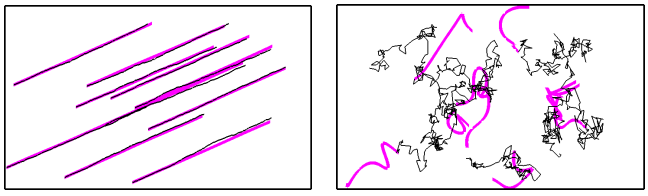
is more or less comparable.

- If the temporal density is high, the error from the approximation of the spanning trajectories becomes negligible compared to the error at the endpoints.
- For extreme sparsity in time, the Bézier curves approximate the trajectories better in the double gyre dataset because they can better encode their highly bent, complicated shapes. In contrast to that, the pathlines in the ABC dataset are short and leave the domain quickly on rather straight trajectories. Here, the polygonal chains perform a little better than the Bézier curves.
- There is usually only a very small difference between the C^0 and the C^1 versions of the representations. There is one exception. The smoothened Bézier curves tend to have a bigger error in the case of $N_t = 2$ in the double gyre dataset. The reason is that for $N_t = 2$, the derivatives sometimes differ greatly. Smoothing out the adjacent control points can lead to an overshoot of the pathlines outside of the domain.

All in all, the approximation error is dominated by the interpolation and the patching of the spanning trajectories in space, which all representations have in common. Therefore, the expected gain in accuracy from using the Bézier curves did not show in our experiments, except for the case of very long and curvy spanning trajectories and in this case, the smoothing to get C^1 curves significantly increases the error. We expect the difference in the asymptotic behavior to show in the case of much higher spacial density, though.

6.4 Smoothness

The visual results for the first two flow fields do not differ gravely for the different curve representations. All trajectory representations perform very well on the distorted laminar flow and very poorly on the turbulent one. An impression of the approximation using the C^1 composite Bézier curves can be seen in Figure 5.

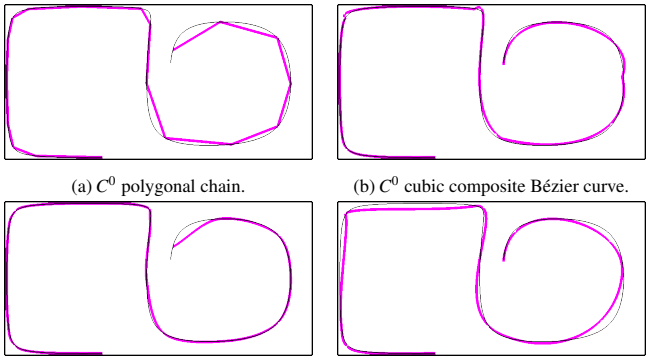


(a) Distorted laminar flow. (b) Turbulent flow.

Figure 5: Randomly seeded pathlines. The integrated lines are drawn in dark grey, the interpolated lines using the C^1 composite Bézier curves with $N_t = 2$ in pink.

Figure 6 shows one randomly started pathline in the double gyre dataset. In all images $N_x = 64 \times 32$ spanning streamlines were evenly distributed. For the polygonal chains, we partitioned the total integration time into $N_t^S = 18$ time steps, for the cubic Bézier curves into only $N_t^B = 6$ to even the storage space. A zoom into the top left corner is depicted in Figure 7 for the Bézier curves. Here, the smoothing process and the resulting overshoot of the C^1 curve can be observed due to the explicit depiction of the control points.

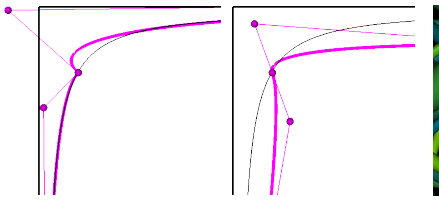
In Figure 8, we show randomly seeded, integrated pathlines in the ABC dataset. The same pathlines interpolated using the different trajectory representations can be found in Figure 9. In all interpolation schemes, $N_x = 64^3$ spanning streamlines were used. Again, we used $N_t^S = 18$ time slices for the linear trajectories and $N_t^B = 6$ for the cubic Bézier curves. The color map in Figure 8 encodes the time parameter. It is chosen to represent a flower that starts in last year's grey green, then grows fresh juicy green sprouts, and finally blossoms in pink. It suffices many requirements on a



(a) C^0 polygonal chain. (b) C^0 cubic composite Bézier curve.

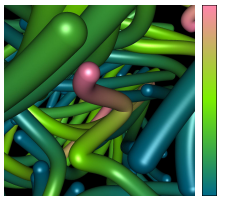
(c) C^1 cubic Hermite spline. (d) C^1 cubic composite Bézier curve.

Figure 6: A randomly seeded pathline in the double gyre dataset. The integrated line is drawn in dark grey, the interpolated line in pink using the different curve representations.



(a) C^0. (b) C^1.

Figure 7: Zoom into Figure 6 for the case of the cubic composite Bézier curve representations. The integrated line is drawn in dark grey, the interpolated line in pink, the thin pink line connects the control points.



(a) (b)

Figure 8: Randomly seeded pathlines in the ABC flow from integration. The color map represents time.

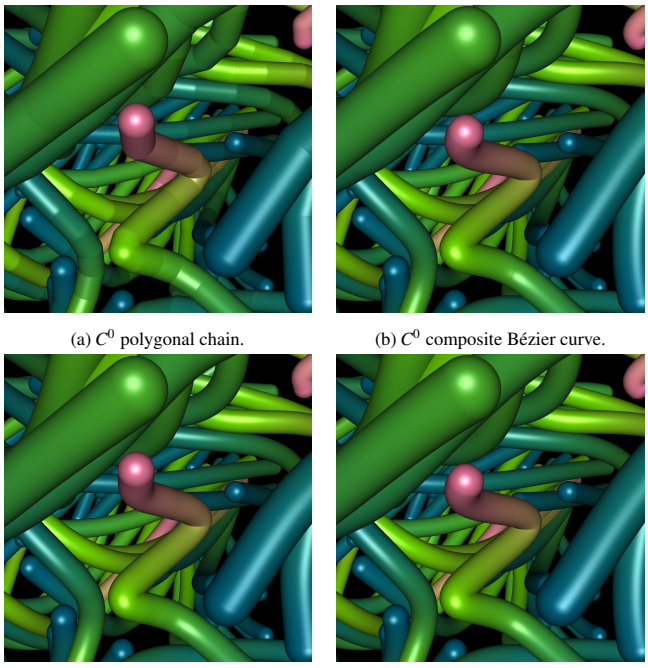
good color map, [23]. It has a good perceptual resolution because it uses three colors, but low differing luminance to not interfere with the 3D shading, is robust with respect to vision deficiencies, and given the interpretation, the order of the colors is intuitively the same for everyone.

Figure 10 is an example to demonstrate that each of the the trajectory representations can be used for any kind of advection based visualization method. On the left, the stream surface is calculated using Runge Kutta and on the right using the interpolation of the spanning trajectories with $N_x = 64, N_t = 2$ and the C^1 composite Bézier curve.

All in all, the C^1 representations give smooth pathlines. We should keep in mind that the round shapes in the Bézier curves correspond to real shapes in the high resolution simulation, while the shape of the spline is only a heuristic. While the spline has problems to approximate the pathlines at their first and the last segments because of the lack of information at the ends (Figures 6(c), 9(c)), the C^1 Bézier curves are more likely to overshoot (Figures 6(d), 7). Both C^0 representations result in non-smooth curves with sharp edges. For a deeper evaluation and comparison of the esthetic quality of each representation, a user study is required.

7 Conclusion

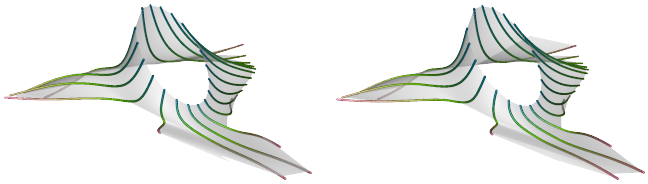
We have compared four methods to construct pathlines using parametric Lagrangian flow field representations. Theoretically, the composite cubic Bézier curves showed to have a smaller approximation error for small time intervals, but in our experiments, the error from the interpolation in space is dominant, which all trajectory representations share. There was no accuracy gain from us-



(a) C^0 polygonal chain. (b) C^0 composite Bézier curve.

(c) C^1 cubic Hermite spline. (d) C^1 composite Bézier curve.

Figure 9: Randomly seeded pathlines in the ABC dataset interpolated using the different curve representations.



(a) Integration. (b) C^1 composite Bézier curve.

Figure 10: Stream surface in the ABC dataset.

ing the Bézier curves instead of linear interpolation. The spacial density has to be much higher than the temporal one to notice the different asymptotic behavior of the two approaches. The C^1 cubic composite Bézier curves and the cubic Hermite splines resulted in smoother curves than their C^0 counterparts.

References

[1] A. Agranovsky, D. Camp, C. Garth, E. Bethel, K. Joy, and H. Childs. Improved post hoc flow analysis via lagrangian representations. In *Large Data Analysis and Visualization (LDAV), 2014 IEEE 4th Symposium on*, pages 67–75, Nov 2014.

[2] A. Agranovsky, C. Garth, and K. I. Joy. Extracting flow structures using sparse particles. In *Proceedings of Vision, Modeling, and Visualization*, pages 153–160, 2011.

[3] A. Brambilla, R. Carnecky, R. Peikert, I. Viola, and H. Hauser. Illustrative Flow Visualization: State of the Art, Trends and Challenges. *EG 2012*, State of the Art Reports:75–94, 2012.

[4] E. Catmull and R. Rom. A class of local interpolating splines. *Computer Aided GeometricDesign*, pages 317–326, 1974.

[5] J. Chandler, H. Obermaier, and K. Joy. Interpolation-based pathline tracing in particle-based flow visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 21(1):68–80, Jan 2015.

[6] C.-M. Chen, A. Biswas, and H.-W. Shen. Uncertainty Modeling and Error Reduction for Pathline Computation in Time-varying Flow Fields. In *IEEE Pacific Visualization Symposium (PacificVis)*, 2015.

[7] H. Childs, D. Pugmire, S. Ahern, B. Whitlock, M. Howison, Prabhat, G. Weber, and E. W. Bethel. Extreme Scaling of Production Visualization Software on Diverse Architectures. *IEEE Computer Graphics and Applications (CG&A)*, 30(3):22–31, May/June 2010.

[8] G. Farin. *Curves and Surfaces for CAGD: A Practical Guide*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition, 2002.

[9] O. Forster. *Analysis 2*. Springer Spektrum, Wiesbaden, Germany, 2013.

[10] C. Garth, F. Gerhardt, X. Tricoche, and H. Hans. Efficient computation and visualization of coherent structures in fluid flow applications. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1464–1471, Nov. 2007.

[11] C. Garth, G. Li, X. Tricoche, C. Hansen, and H. Hagen. Visualizationof coherent structures in transient 2d flows. In *Proceedings of Topology-Based Methods in Visualization*, 2007.

[12] D. F. Griffiths and D. J. Higham. *Numerical Methods for Ordinary Differential Equations: Initial Value Problems*. Springer, London, UK, 2010.

[13] G. Haller. Lagrangian Coherent Structures from Approximate Velocity Data. *Physics of Fluids*, 14:1851–1861, 2002.

[14] G. Haller and G. Yuan. Lagrangian coherent structures and mixing in two-dimensional turbulence. *Phys. D*, 147(3-4):352–370, Dec. 2000.

[15] M. Hlawatsch, F. Sadlo, and D. Weiskopf. Hierarchical line integration. *IEEE Transactions on Visualization and Computer Graphics*, 17(8):1148–1163, 2011.

[16] J. Hultquist. Interactive Numerical Flow Visualization. *Computing Systems in Engineering*, 1(2-4):349–353, 1990.

[17] B. Jobard, G. Erlebacker, and H. Hussaini. Lagrangian-Eulerian advection of noise and dye textures for unsteady flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):211–222, 2002.

[18] D. H. U. Kochanek and R. H. Bartels. Interpolating splines with local tension, continuity, and bias control. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '84, pages 33–41, New York, NY, USA, 1984. ACM.

[19] R. S. Laramee, H. Hauser, H. Doleisch, B. Vrolijk, F. H. Post, and D. Weiskopf. The State of the Art in Flow Visualization: Dense and Texture-Based Techniques. *Computer Graphics Forum*, 23:2004, 2004.

[20] R. S. Laramee, H. Hauser, L. Zhao, and F. H. Post. Topology-Based Flow Visualization, The State of the Art. In *Topology-based Methods in Visualization*, pages 1–19, 2007.

[21] T. McLoughlin, R. S. Laramee, R. Peikert, F. H. Post, and M. Chen. Over Two Decades of Integration-Based, Geometric Flow Visualization. In *EG 2009 - State of the Art Reports*, pages 73–92, 2009.

[22] J. J. Monaghan. Smoothed particle hydrodynamics. *Reports on Progress in Physics*, 68(8):1703, 2005.

[23] K. Moreland. Diverging color maps for scientific visualization. In *Advances in Visual Computing*, pages 92–103. Springer, 2009.

[24] A. Pobitzer, R. Peikert, R. Fuchs, B. Schindler, A. Kuhn, H. Theisel, K. Matkovic, and H. Hauser. The State of the Art in Topology-based Visualization of Unsteady Flow. *Computer Graphics Forum*, 30(6):1789–1811, September 2011.

[25] F. Sadlo and R. Peikert. Efficient visualization of lagrangian coherent structures by filtered amr ridge extraction. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1456–1463, 2007.

[26] T. Salzbrunn, C. Garth, G. Scheuermann, and J. Meyer. Pathline predicates and unsteady flow structures. *The Visual Computer*, 24(12):1039–1051, 2008.

[27] T. Salzbrunn, H. Jänicke, T. Wischgoll, and G. Scheuermann. The State of the Art in Flow Visualization: Partition-based Techniques. In *Simulation and Visualization 2008 Proceedings*, 2008.

[28] T. Salzbrunn and G. Scheuermann. Streamline predicates. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1601–1612, 2006.

[29] M. Schatzmann. *Numerical Analysis: A Mathematical Introduction*. Oxford University Press, New York, USA, 2002.

[30] A. J. Smits and T. T. Lim. *Flow Visualization Techniques and Examples*. Imperil College Press, London, UK, 2000.