Implicit Ray Casting of the Parallel Vectors Operator

Ramon Witschi and Tobias Günther
Department of Computer Science, ETH Zurich

ABSTRACT

Feature extraction is an essential aspect of scientific data analysis, as it allows for a data reduction onto relevant structures. The extraction of such features from scalar and vector fields, however, can be computationally expensive and numerically challenging. In this paper, we concentrate on 3D line features in vector fields that are defined by the parallel vectors operator. Common examples are vortex corelines and hyperbolic trajectories, i.e., lines around which particles are rotating, or from which particles are repelled and attracted locally the strongest. In our work, we use a GPU volume rendering framework to calculate the lines on-the-fly via a parallel vectors implementation in the volume rendering kernels. We achieve real-time performance for the feature curve extraction, which enables interactive filtering and parameter adjustment.

Keywords: Scientific visualization, vortex extraction, parallel vectors

1 INTRODUCTION

Vector fields can contain numerous features, such as vortices [8, 28, 30] and hyperbolic trajectories [12, 16, 24, 27]. The study of such features remains important in many application areas, such as computational fluid dynamics, medicine and various engineering disciplines. For the definition of line features, the parallel vectors operator [23] proved to be a versatile formalism, which identifies locations at which two vector fields are parallel. Past research in this area revealed two main classes of extraction approaches: local techniques that decide per voxel if a feature curve exists [18, 23, 30], and integration-based techniques that trace out the curves by integration in a derived vector field [1, 3, 29, 31, 34]. However, the precise numerical extraction of those curves is often time-consuming, which makes adjustments of the visualization a posteriori cumbersome. For this reason, feature curves are typically first extracted in a preprocess and are afterwards filtered and visualized.

In this paper, we follow a different approach. We devise an efficient preview rendering technique, which can be used during parameter exploration of the feature definition. This is especially time-saving when the underlying fields are parameter-dependent, for instance reference frame parameters [2, 11] or the size of finite-sized particles [7]. Afterwards, explicit line geometry can be extracted either with a particle-based approach [19] or with traditional grid-based subdivision [23]. For efficient preview rendering, we implement a parallel vectors ray caster directly on the GPU by casting the feature curves implicitly. Rather than phrasing the parallel vector line computation as an extremal line extraction as done by Kindlmann et al. [19], we phrase it as a root-finding problem. Our lines are implicitly represented by spheres or cylinders in order to resemble tubes, and the GPU-based real-time ray tracing is capable of varying tube radius, color and transparency for shading based on local properties. The key to an efficient implicit ray casting of parallel vector features is two-fold. First, an empty cell skipping is required to avoid work in voxels that cannot contain parallel vectors solutions. To do so, we transform the cross product of the trilinearly interpolated vector fields into Bézier-Bernstein basis and use the convex hull property to discard voxels for which roots cannot exist in the cross-product components. Second, we need an efficient method to determine a nearby parallel vectors feature curve in order to intersect the view ray implicitly with the curve representation. For this, we apply an efficient sectional Newton descent solver. Our contributions are

- the first implicit ray caster of parallel vectors feature curves that reaches real-time performance for interactive parameter exploration and preview,
- two acceleration methods that are tailored to the extraction of parallel vector solutions in a direct volume renderer, namely empty-space skipping through tensor product culling and a sectional Newton descent for implicit rendering of the feature curves.

The novelty of the method lies in the application of tensor product culling and the sectional Newton descent in an interactive ray caster to implicitly render parallel vectors solutions.

2 RELATED WORK

2.1 Parallel Vectors Operator

The parallel vectors (PV) operator is a versatile operator that was introduced by Peikert and Roth [23] in order to describe a number of feature curves in scalar and vector fields. Given two vector fields $\mathbf{v}(\mathbf{x}), \mathbf{w}(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^n$, the PV operator is defined as follows

$$\mathbf{v} \parallel \mathbf{w} = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{v}(\mathbf{x}) = \mathbf{0}\} \cup \{\mathbf{x} \in \mathbb{R}^n \mid \exists \lambda \in \mathbb{R} : \mathbf{v}(\mathbf{x}) = \lambda \mathbf{w}(\mathbf{x})\}$$

In other words, given two vector fields $\mathbf{v}(\mathbf{x})$ and $\mathbf{w}(\mathbf{x})$, the parallel vectors operator returns a set of points, representing all locations at which those two vector fields are parallel, i.e., linearly dependent. An equivalent formulation in three dimensions is given by localization of the roots of the cross product of $\mathbf{v}$ and $\mathbf{w}$, i.e.:

$$\mathbf{v} \parallel \mathbf{w} \;\Leftrightarrow\; \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{v}(\mathbf{x}) \times \mathbf{w}(\mathbf{x}) = \mathbf{0}\} \tag{1}$$

A number of algorithms have been proposed to extract the features defined by the PV operator, which can be classified into local and integration-based techniques. Among the local methods, Peikert and Roth [23] showed that the solution on the faces of piece-wise linear vector fields is an eigenvector problem, as used for instance by Gerrits et al. [5]. For trilinear vector fields, Newton-Raphson iterations can be applied on each cell face to descend from an initial solution. Ju et al. [17] proposed a parity test to determine the number of parallel vector solutions on the face of a cell. In higher dimensions [9, 21] tensor products have been used to recursively subdivide cells, narrowing down the location of the parallel vectors solution. Recently, a high-dimensional dependent vectors operator was introduced by Hofmann and Sadlo [14]. Among the integration-based methods, feature flow fields [32] have been used to track parallel vector solutions [31]. Van Gelder and Pang [34] extended the above approach, accounting for a direction change of the PV solution. Weinkauf et al. [36] proposed a stabilization to reduce the integration error by adding a vector field that guides back to the parallel vectors solution. Pagot et al. [22] searched for parallel vector lines in higher-order

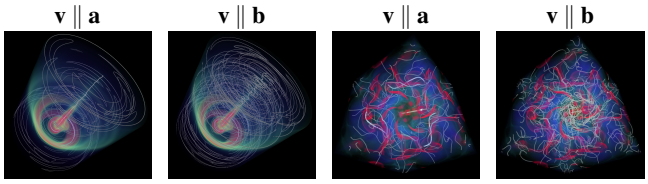


Figure 1: *Comparison of parallel vectors solutions to* $\mathbf{v} \parallel \mathbf{a}$ *and to* $\mathbf{v} \parallel \mathbf{b}$*. Here, shown with direct volume rendering of vorticity for the* SWIRLING JET *(left) and the* BORROMEAN RINGS *(right).*

vector fields. Gerrits et al. [5] searched for the approximate parallel vectors solution in an ensemble of flow fields, for which they found the lines at which the mean field and the eigendirection with highest variance are aligned. The approximate PV solution of an entire ensemble is thereby reduced to a single standard PV problem. Hofmann and Sadlo [15] considered an n-dimensional vector field with k additional vector fields. They searched for the k-dimensional space, where all n-dimensional vector fields are linearly dependent, which is expressed as roots of the wedge product. For this, they iterate the $(n-k)$-faces of the n-dimensional grid and apply Gauss-Newton iterations to find one of the isolated intersection points of the k-dimensional solution space. The isolated points are later recursively connected to form manifolds. Hofmann and Sadlo formulated a general feature extraction algorithm for high-dimensional vector fields, while we accelerate a special case: the parallel vectors operator for $n = 3$ and $k = 1$. In contrast to the approaches above, we extract the parallel vectors solutions implicitly and along a ray rather than on the faces of a discretization (tetrahedra or voxels).

In the same spirit, Kindlmann et al. [19] proposed an implicit ray casting method of extremal lines and surfaces that uses a standard Newton descent scheme. Since they described the parallel vector solution as extremal line of $P(\mathbf{x})$, along which the dot product of the two normalized vector fields is $+1$ or -1:

$$P(\mathbf{x}) = \frac{\mathbf{v}(\mathbf{x})}{\|\mathbf{v}(\mathbf{x})\|} \cdot \frac{\mathbf{w}(\mathbf{x})}{\|\mathbf{w}(\mathbf{x})\|} \tag{2}$$

their approach needs second-order derivatives of $\mathbf{v}/\|\mathbf{v}\|$ and $\mathbf{w}/\|\mathbf{w}\|$. The division by the norm can cause problems, when $\|\mathbf{v}\| = 0$ or $\|\mathbf{w}\| = 0$. For example, in case of vortex coreline extraction in 2D space-time, Sujudi-Haimes [30] is directly Galilean-invariant. However, in this case, the PV solution is equivalently found as location with zero acceleration [8], i.e., $\|\mathbf{w}\| = 0$. Our method, on the other hand, seeks roots of the cross product of the two vector fields $\mathbf{v}$ and $\mathbf{w}$, which only requires first-order derivatives of $\mathbf{v}$ and $\mathbf{w}$.

2.2 Feature Lines in Scientific Visualization

The parallel vectors operator lends itself to define many feature lines that are of interest in scientific visualization problems. We refer to Kindlmann et al. [19] for the extraction of extremal features and to Peikert and Roth [23] for an overview of feature curves expressed by the parallel vectors operator. Given a steady vector field $\mathbf{v}(\mathbf{x})$, three derived quantities are needed to define feature curves. With $\nabla = \frac{\partial}{\partial \mathbf{x}}$ being the nabla operator, we use Jacobian $\mathbf{J}(\mathbf{x})$, acceleration $\mathbf{a}(\mathbf{x})$ and jerk vector $\mathbf{b}(\mathbf{x})$:

$$\mathbf{J}(\mathbf{x}) = \nabla \mathbf{v}(\mathbf{x}), \quad \mathbf{a}(\mathbf{x}) = \mathbf{J}(\mathbf{x}) \cdot \mathbf{v}(\mathbf{x}), \quad \mathbf{b}(\mathbf{x}) = \nabla \mathbf{a}(\mathbf{x}) \cdot \mathbf{v}(\mathbf{x}) \tag{3}$$

In this paper, we extract first-order feature curves using $\mathbf{v} \parallel \mathbf{a}$ [27,30] and second-order feature curves using $\mathbf{v} \parallel \mathbf{b}$ [28], as shown in Fig. 1. The feature curve segments are categorized by the eigenvalues of the Jacobian into vortex corelines (eigenvalues are complex) and bifurcation lines (attraction and separation are present) [27,30].

3 REAL-TIME RAY CASTING OF LINE FEATURES

Our goal is to efficiently render the solutions to the parallel vectors operator within a GPU-based volume ray casting framework. Input to our method are the two vector fields $\mathbf{v}(\mathbf{x})$ and $\mathbf{w}(\mathbf{x})$, which are both discretized onto a regular grid. The output of our approach is an image, in which the solutions $\mathbf{v}(\mathbf{x}) \parallel \mathbf{w}(\mathbf{x})$ are visually represented by many spheres or cylinders in order to resemble tubes. In order to achieve interactivity, we employ two acceleration strategies: we pre-compute a filter mask for empty space skipping and we use a sectional Newton descent to quickly descend to a PV solution. Fig. 2 illustrates our pipeline steps in the steady DELTA WING flow. In (a), streamlines of the input vector fields $\mathbf{v}$ and $\mathbf{w}$ are shown. The pre-computed filter mask is visible in (b). Our solution is compared to an offline method in (c). Finally, (d) shows our solution along with a volume rendering of vorticity and selected streamlines. We refer to the additional material for pseudocode of the various steps. In the following, we discuss the implicit ray casting and its acceleration strategies in more detail.

3.1 Filter Mask for Empty Space Skipping

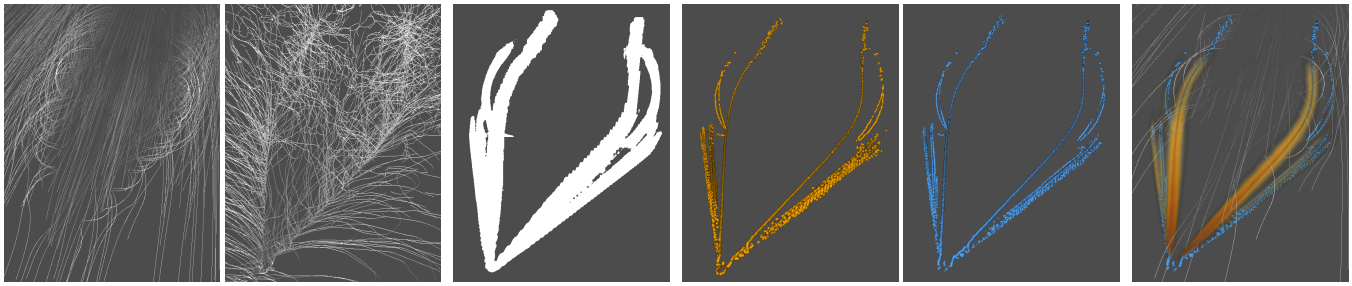
The runtime of our algorithm is mainly determined by the search for a nearby parallel vectors solution inside the volume rendering kernels. In order to avoid unnecessary work, we pre-compute a filter field $s(\mathbf{x})$ that allows us to identify voxels in which a PV solution is guaranteed to not exist. These voxels can be skipped during ray marching. We assume that the vector fields $\mathbf{v}(\mathbf{x})$ and $\mathbf{w}(\mathbf{x})$ are both trilinearly interpolated from a grid. Since PV solutions are curves, we know that the curves will intersect the faces of the voxels. In order to determine whether a solution passes through a voxel, it is sufficient for us to check whether any of the faces contains a PV solution. In principle, it is possible that a PV line exists inside a voxel and never leaves it. We will ignore those cases, since we are interested in feature curves larger than a voxel. On the face of each voxel, the vector fields $\mathbf{v}(\mathbf{x})$ and $\mathbf{w}(\mathbf{x})$ are interpolated bilinearly. Parallel vector solutions are locations at which the cross product vanishes:

$$\mathbf{f}(x,y) = \mathbf{v}(x,y) \times \mathbf{w}(x,y) = \begin{pmatrix} f_1(x,y) \\ f_2(x,y) \\ f_3(x,y) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{4}$$

Finding the locations (x,y) therefore becomes a root-finding problem. A simple way to test whether the scalar components $f_k(x,y)$ can contain a root for any $x,y \in [0,1]$ is to transform the scalar fields into a tensor product representation [21]. Since a tensor product surface always stays within the convex hull of its control polygon, it is sufficient to test whether all Bézier control points are positive or negative. With the Bernstein basis functions $B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$, we convert each component of Eq. (4) into a bi-quadratic tensor product surface:

$$f_k(x,y) = \sum_{i=0}^{2} \sum_{j=0}^{2} B_i^n(x) \cdot B_j^n(y) \cdot \mathbf{b}_{i,j}^k \tag{5}$$

Note that a bi-quadratic representation is needed, since the cross product of two bilinear fields in Eq. (4) is bi-quadratic. The Bézier control polygons of the three scalar components f_1, f_2 and f_3 are illustrated in Fig. 3. The above PV test can only tell whether a PV solution cannot exist. In order to find the precise location, the quad would have to be recursively subdivided. We use this recursive approach to obtain the ground truth PV locations or when the user wishes to extract an explicit parallel vectors solution. For our filter $s(\mathbf{x})$, we do not perform the subdivisions and instead compute a binary flag that determines whether a solution may exist inside a voxel. Since we are ultimately placing spheres or cylinders at PV lines, we grow the masked regions by the size of the maximum



(a) *Input vector fields* **v** *(left) and* **w** *(right)* **(b)** *Filter mask* **(c)** *Ground truth (left) and ours (right)* **(d)** *with vorticity*

Figure 2: *Overview of our implicit parallel vectors ray casting pipeline. (a) Input to our ray casting module are the two vector fields* **v** *and* **w**. *(b) We compute a filter mask that allows us to do empty-space skipping. (c) Our ray caster and an offline computation method produce the same result. (d) The implicitly ray casted PV curves are shown interactively together with a context volume rendering and streamlines.*

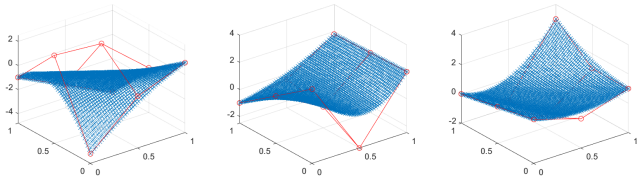


Figure 3: *Visualization of the three cross product components* $f_1(x,y)$, $f_2(x,y)$ *and* $f_3(x,y)$ *(blue) of two bilinearly interpolated vector fields, and their corresponding Bézier control polygons (red). If all control points are above or below zero for any of the components, the blue surface cannot cross zero, i.e., there is no parallel vectors solution.*

radius that we are going to apply. An example of a filter mask is shown in Fig. 2b.

3.2 Ray Marching

We integrate our implicit parallel vectors ray caster in a standard direct volume renderer [10] with early ray termination. We use a GPU ray casting framework that parallelizes the computation across the pixels on the screen and apply a step size of half a voxel to avoid aliasing. At a given location $\mathbf{x}$ along the ray, we first perform a lookup from the filter field $s(\mathbf{x})$ to determine whether the parallel vectors search can be skipped, either because no parallel vectors solution can exist or because a custom filter does not apply. If $\mathbf{x}$ passes the filter criterion, it is treated as a seed point for a root-finding scheme, which potentially returns the location of a root $\mathbf{x}^*$. If a root has been found, the line geometry has to be spanned. A common approach is to use cylinders [13], for which we need to determine the orientation vector $\mathbf{t}$ in order to align the cylinder with the direction of the feature line. We used the feature flow field [32] of Theisel et al. [31] which they proposed to trace parallel vector solutions, i.e., they constructed a derived vector field that is tangential to the feature curve by the fact that it must be perpendicular to the gradients of the components of $\mathbf{f} = \mathbf{v} \times \mathbf{w}$ at root $\mathbf{x}^*$. The radius of the cylinder is fixed globally or can be mapped via transfer function from a scalar field. Note that the root-finding can be terminated if the search is taking us too far away from the ray and a cylinder intersection cannot exist. If a cylinder intersection is found, the fragment is shaded. Optionally, transparency is assigned via transfer function. The quality of the visualization can be traded for performance by varying the ray marching step size, as shown in the additional material.

3.3 Locating Feature Curves

The most performance-critical component of our algorithm is the fast localization of a nearby parallel vectors solution. With Eq. (1), we have seen that the search for a parallel vectors solution is a root-finding problem in 3D, i.e.:

$$\mathbf{v}(\mathbf{x}) \times \mathbf{w}(\mathbf{x}) = \mathbf{f}(\mathbf{x}) = \begin{pmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ f_3(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{6}$$

In contrast to the previous sections, we are now searching for a solution in 3D, instead of a solution on the 2D face of a voxel.

In general, the roots of a multi-variate function can be found with the Newton-Raphson algorithm [26], which requires the computation of the pseudoinverse of $\nabla\mathbf{f}(\mathbf{x})$, since this matrix becomes singular on the PV solution [34]. In order to avoid the computationally expensive calculation of the pseudoinverse of a 3×3 matrix, we use the following simpler algorithm.

Sectional Newton Descent. Schindler et al. [29] proposed a predictor-corrector method to trace a parallel vectors solution, which includes a corrector part that descends onto a nearby PV line that we use as follows. The Jacobian $\nabla\mathbf{f}$ of the cross-product $\mathbf{f}$ in Eq. (6) tells us by first-order Taylor approximation how $\mathbf{f}$ changes when stepping in a certain direction $\mathbf{d} = \mathbf{x}_{i+1} - \mathbf{x}_i$:

$$\mathbf{f}(\mathbf{x}_{i+1}) \approx \mathbf{f}(\mathbf{x}_i) + \nabla\mathbf{f}(\mathbf{x}_i) \cdot (\mathbf{x}_{i+1} - \mathbf{x}_i) \tag{7}$$

$$= \mathbf{f}(\mathbf{x}_i) + \nabla\mathbf{f}(\mathbf{x}_i) \cdot \mathbf{d} \tag{8}$$

Our goal is to find roots of $\mathbf{f}$. Requesting that the next value $\mathbf{f}(\mathbf{x}_{i+1})$ vanishes to zero, i.e., $\mathbf{f}(\mathbf{x}_{i+1}) = \mathbf{0}$, gives:

$$\nabla\mathbf{f}(\mathbf{x}_i) \cdot \mathbf{d} = -\mathbf{f}(\mathbf{x}_i) \tag{9}$$

Since the third component of the cross-product vanishes to zero if the first two components are zero [29] (except for the special case of $\mathbf{v}(\mathbf{x}) \neq \mathbf{0}$, $\mathbf{w}(\mathbf{x}) \neq \mathbf{0}$ and $v_3 = w_3 = 0$ [29], see additional material), it is sufficient to consider only two of the components. To maximize the improvement we select the two components f_k, f_l with largest magnitude of the cross product of the gradients:

$$k,l = \underset{k,l \in \{0,1,2\}}{\arg\max} \; \|\nabla f_k(\mathbf{x}_i) \times \nabla f_l(\mathbf{x}_i)\|_2 \;, \quad \text{s.t. } k \neq l \tag{10}$$

which maximizes the gradient norms and the angle between the gradients, since $\|\nabla f_k(\mathbf{x}_i) \times \nabla f_l(\mathbf{x}_i)\|_2 = \|\nabla f_k(\mathbf{x}_i)\|_2 \cdot \|\nabla f_l(\mathbf{x}_i)\|_2 \cdot \sin\theta_{k,l}$. Maximizing the angle as well avoids the selection of linearly dependent gradients. We can expect the biggest change in direction $\mathbf{d}$ to be a linear combination of the two gradients maximizing Eq. (10). We can solve for $\mathbf{d}$ directly using

$$\mathbf{d} = -\left(\nabla f_k(\mathbf{x}_i),\, \nabla f_l(\mathbf{x}_i)\right) \cdot \mathbf{A}^{-1} \cdot \begin{pmatrix} f_k(\mathbf{x}_i) \\ f_l(\mathbf{x}_i) \end{pmatrix} \tag{11}$$

Data Set	Grid Resolution	PV baseline in sec.	pre-compute Filter $s(\mathbf{x})$ in sec.	Ours, far view in fps. w/ Filter	Ours, far view in fps. w/o Filter	Ours, close-up in fps. w/ Filter	Ours, close-up in fps. w/o Filter
Stuart Vortex	$128 \times 128 \times 128$	0.70	0.07	11.71	2.60	16.15	3.84
Borromean Rings	$128 \times 128 \times 128$	0.87	0.07	10.91	4.17	7.85	2.51
Cylinder	$380 \times 47 \times 100$	1.52	0.18	20.73	7.89	13.17	3.98
Swirling Jet	$201 \times 221 \times 201$	3.28	0.32	6.33	2.34	5.19	1.48
Delta Wing	$250 \times 125 \times 100$	0.89	0.18	11.03	10.56	11.63	10.21

Table 1: *For each data set: grid resolution, time to compute a baseline with tensor product subdivision (until 10^{-5} residual), time for the one-time filter field precomputation, frames per second (fps) for our implicit ray-marching using the NVIDIA IndeX CUDA kernel implementation with a ray traversal step size of half a voxel, 1000×1000 resolution, and 10 iterations of sectional Newton descent, with (w/) and without (w/o) filter fields. All timings for $\mathbf{v} \parallel \mathbf{a}$ without volume rendering or LIC.*

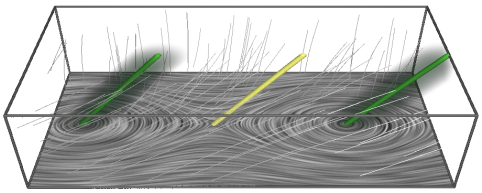


Figure 4: *Visualization of the 2D* STUART *vortex in 2D space-time, i.e., time is mapped to the vertical spatial dimension.*

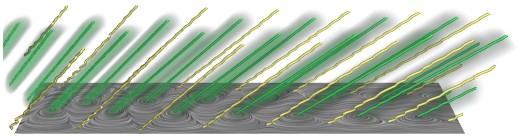


Figure 5: *Visualization of PV solutions in the* CYLINDER *flow.*

where $\mathbf{A}$ is a symmetric matrix

$$\mathbf{A} = \begin{pmatrix} \|\nabla f_k(\mathbf{x}_i)\|_2^2 & \nabla f_k(\mathbf{x}_i) \cdot \nabla f_l(\mathbf{x}_i) \\ \nabla f_l(\mathbf{x}_i) \cdot \nabla f_k(\mathbf{x}_i) & \|\nabla f_l(\mathbf{x}_i)\|_2^2 \end{pmatrix} \tag{12}$$

Derivations are in the additional material. For Eq. (11), we symbolically invert the positive semi-definite 2×2 matrix.

4 RESULT

Next, we apply our method to a number of vector fields. Note that while the parallel vectors operator is defined for 3D steady flows, it can be applied to unsteady flows with certain assumptions [6] or after certain transformations [2, 11]. Thus, we can apply our approach to 3D steady flows (in space) and 2D unsteady flows (in space-time).

4.1 Datasets

We use a 2D version [6] of a uniformly-translating Stuart vortex [35]. In Fig. 4, we show the vortex corelines (green) and the bifurcation line (yellow) in the domain $[-6,3] \times [-2,2] \times [1,3]$. The swirling strength, i.e., the imaginary part of the Jacobian, is visualized with direct volume rendering, indicating that swirling motion is present near the vortex corelines. In addition, pathlines are shown in 2D space-time, and a LIC slice is shown, for which the unit translation was subtracted from the u component to reveal the critical points. The parallel vector lines pass through critical points of the LIC slice.

In Fig. 5, we study a numerically simulated unsteady 2D vector field [6, 25]. Since vortices move with nearly constant speed, we assume Galilean invariance extracting vortex corelines and bifurcation lines in 2D space-time with the criterion $\mathbf{v} \parallel \mathbf{a}$.

The DELTA WING [33] was provided by Markus Rütten. To filter the vortex corelines from the remaining PV solution, we apply a swirling strength filter of 0.1. Results are shown in Fig. 2, where the inputs, filter mask, a ground truth comparison and a context visualization of vorticity are found.

All previous examples only applied the first-order vortex criterion $\mathbf{v} \parallel \mathbf{a}$. In the following, we apply our implicit parallel vectors ray caster to the numerically more challenging criterion $\mathbf{v} \parallel \mathbf{b}$. For this, we visualize all PV solution in a magnetic field [4]. For comparison, we show the solutions to $\mathbf{v} \parallel \mathbf{a}$ and the solutions to $\mathbf{v} \parallel \mathbf{b}$ in Fig. 1 (bottom) together with a volume rendering of the vorticity.

Our last example contains a 3D unsteady swirling jet flow that undergoes a vortex breakdown [20]. In this flow, we visualize parallel vectors solutions of a single time slice, i.e., we visualize streamline-oriented behavior and not pathline-oriented behavior. We demonstrate the unfiltered solutions to $\mathbf{v} \parallel \mathbf{a}$ and $\mathbf{v} \parallel \mathbf{b}$ in Fig. 1 (top).

4.2 Performance

We tested our method on an Intel Core i7-8700k, with 6 physical cores clocked at 3.7Ghz, 48GB RAM and an RTX2080TI with 11GB VRAM. In principle, our approach can be implemented in any existing volume ray marcher, see Alg. 2 in additional material. We implemented the method with floating point precision both in Nvidia Index using CUDA kernels, as well as in Direct3D using pixel shaders. An OpenGL implementation would be likewise possible. All performance measurements in Table 1 are reported for the CUDA implementation. Our method includes a pre-processing step to determine the filter mask for empty space skipping, for which the computation time is listed. Across all examples, we obtained a frame rate of at least 6.3*fps* when looking from outside at the domain, and at least 5.1*fps* when moving into the domain. Without empty space skipping, the frame rate dropped in the worst case to 1.4*fps*. The worst performance occurred in the SWIRLING JET due to its dense coverage with parallel vectors solutions. In other vector fields, the empty space skipping could be applied more often, since the parallel vectors solutions were more sparse, resulting in a better performance. The scaling for varying viewport resolutions is shown in the additional material, demonstrating interactive performance.

5 CONCLUSION

In flow visualization, we are frequently interested in line features, which are usually extracted and visualized in two separate steps. In this paper, we integrated the feature extraction into the rendering algorithm in an implicit way. To this end, we implemented the parallel vectors operator inside a direct volume renderer, which is accelerated by the NVIDIA IndeX API. In order to accelerate the computation, we applied two optimization strategies: the utilization of a filter field that analyses the tensor product representation of the cross product components of the parallel vectors, and a sectional Newton descent algorithm that quickly identifies a nearby parallel vectors solution, allowing us to implicitly create a geometry that the view ray is intersected with. Implicit methods have the advantage that they only compute the parallel vectors problem, where the structures are visible. In the future, we plan to investigate adaptive methods to march through empty spaces quicker.

ACKNOWLEDGMENTS

We thank Alexander Kuhn and the IndeX team at Nvidia Berlin for the fruitful discussions. This work was supported by the Swiss National Science Foundation (SNSF) Ambizione grant no. PZ00P2_180114.

REFERENCES

[1] R. Bader, M. Sprenger, N. Ban, S. Rüdisühli, C. Schär, and T. Günther. Extraction and visual analysis of potential vorticity banners around the alps. *IEEE Transactions on Visualization and Computer Graphics (Proc. IEEE Scientific Visualization 2019)*, 26:256–259, 2020. doi: 10.1109/TVCG.2019.2934310

[2] I. Baeza Rojo and T. Günther. Vector field topology of time-dependent flows in a steady reference frame. *IEEE Transactions on Visualization and Computer Graphics (Proc. IEEE Scientific Visualization 2019)*, 26:280–290, 2020. doi: 10.1109/TVCG.2019.2934375

[3] D. Banks and B. Singer. A predictor-corrector technique for visualizing unsteady flow. *Visualization and Computer Graphics, IEEE Transactions on*, 1:151 – 163, 07 1995. doi: 10.1109/2945.468404

[4] S. Candelaresi and A. Brandenburg. Decay of helical and nonhelical magnetic knots. *Phys. Rev. E*, 84:016406, 2011. doi: 10.1103/PhysRevE.84.016406

[5] T. Gerrits, C. Rössl, and H. Theisel. An approximate parallel vectors operator for multiple vector fields. *Computer Graphics Forum (Proc. EuroVis 2018)*, 37(3):315–326, 2018. doi: 10.1111/cgf.13422

[6] T. Günther, M. Gross, and H. Theisel. Generic objective vortices for flow visualization. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 36(4):141:1–141:11, 2017. doi: 10.1145/3072959.3073684

[7] T. Günther and H. Theisel. Vortex cores of inertial particles. *IEEE Transactions on Visualization and Computer Graphics (Proc. IEEE Scientific Visualization 2014)*, 20(12):2535–2544, dec 2014. doi: 10.1109/TVCG.2014.2346415

[8] T. Günther and H. Theisel. The state of the art in vortex extraction. *Computer Graphics Forum*, 37:149–173, 2018. doi: 10.1111/cgf.13319

[9] T. Günther and H. Theisel. Objective vortex corelines of finite-sized objects in fluid flows. *IEEE Transactions on Visualization and Computer Graphics (Proc. IEEE Scientific Visualization 2018)*, 25(1):956–966, 2019. doi: 10.1109/TVCG.2018.2864828

[10] M. Hadwiger, P. Ljung, C. R. Salama, and T. Ropinski. Advanced illumination techniques for GPU-based volume raycasting. In *ACM SIGGRAPH 2009 Courses*, SIGGRAPH '09, pp. 2:1–2:166. ACM, New York, NY, USA, 2009. doi: 10.1145/1667239.1667241

[11] M. Hadwiger, M. Mlejnek, T. Theußl, and P. Rautek. Time-dependent flow seen through approximate observer killing fields. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):1257–1266, Jan 2019. doi: 10.1109/TVCG.2018.2864839

[12] G. Haller. Finding finite-time invariant manifolds in two-dimensional velocity fields. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 10(1):99–108, 2000. doi: 10.1063/1.166479

[13] M. Han, I. Wald, W. Usher, Q. Wu, F. Wang, V. Pascucci, C. D. Hansen, and C. R. Johnson. Ray tracing generalized tube primitives: Method and applications. *Computer Graphics Forum*, 38(3):467–478, 2019. doi: 10.1111/cgf.13703

[14] L. Hofmann and F. Sadlo. The dependent vectors operator. *Computer Graphics Forum*, 38(3):261–272, 2019. doi: 10.1111/cgf.13687

[15] L. Hofmann and F. Sadlo. The dependent vectors operator. *Computer Graphics Forum (Proc. EuroVis 2019)*, 38(3):261–272, 2019. doi: 10.1111/cgf.13687

[16] L. Hofmann and F. Sadlo. Extraction of distinguished hyperbolic trajectories for2d time-dependent vector field topology. *Computer Graphics Forum (Proc. EuroVis)*, p. in print, 2020.

[17] T. Ju, M. Cheng, X. Wang, and Y. Duan. A robust parity test for extracting parallel vectors in 3D. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2526–2534, Dec 2014. doi: 10.1109/TVCG.2014.2346412

[18] M. Kern, T. Hewson, F. Sadlo, R. Westermann, and M. Rautenhaus. Robust detection and visualization of jet-stream core lines in atmospheric flow. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):893–902, 2017. doi: 10.1109/TVCG.2017.2743989

[19] G. Kindlmann, C. Chiw, T. Huynh, A. Gyulassy, J. Reppy, and P.-T. Bremer. Rendering and extracting extremal features in 3D fields. *Computer Graphics Forum*, 37(3):525–536, 2018. doi: 10.1111/cgf.13439

[20] K. Oberleithner, M. Sieber, C. N. Nayeri, C. O. Paschereit, C. Petz, H.-C. Hege, B. R. Noack, and I. Wygnanski. Three-dimensional coherent structures in a swirling jet undergoing vortex breakdown: stability analysis and empirical mode construction. *Journal of Fluid Mechanics*, 679:383–414, 2011. doi: 10.1017/jfm.2011.141

[21] T. Oster, C. Rössl, and H. Theisel. Core lines in 3D second-order tensor fields. *Computer Graphics Forum (Proc. EuroVis)*, 37(3):327–337, 2018. doi: 10.1111/cgf.13423

[22] C. Pagot, D. Osmari, F. Sadlo, D. Weiskopf, T. Ertl, and J. Comba. Efficient parallel vectors feature extraction from higher-order data. *Computer Graphics Forum*, 30(3):751–760, 2011. doi: 10.1111/j.1467-8659.2011.01924.x

[23] R. Peikert and M. Roth. The "parallel vectors" operator - A vector field visualization primitive. In D. S. Ebert, M. H. Gross, and B. Hamann, eds., *IEEE Visualization 1999, 24-29 October 1999, San Francisco, CA, USA, Proceedings*, pp. 263–270. IEEE Computer Society and ACM, 1999. doi: 10.1109/VISUAL.1999.809896

[24] A. E. Perry and M. S. Chong. A description of eddying motions and flow patterns using critical-point concepts. *Annual Review of Fluid Mechanics*, 19(1):125–155, 1987. doi: 10.1146/annurev.fl.19.010187.001013

[25] S. Popinet. Free computational fluid dynamics. *ClusterWorld*, 2(6), 2004.

[26] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes: the art of scientific computing, 3rd Edition*. Cambridge University Press, 2007.

[27] M. Roth. *Automatic extraction of vortex core lines and other line type features for scientific visualization*. PhD thesis, ETH Zurich, 2000. PhD dissertation number 13673. doi: 10.3929/ethz-a-004016407

[28] M. Roth and R. Peikert. A higher-order method for finding vortex core lines. In *Proc. IEEE Visualization*, pp. 143–150, 1998. doi: 10.1109/VISUAL.1998.745296

[29] B. Schindler, R. Fuchs, J. Biddiscombe, and R. Peikert. Predictor-corrector schemes for visualization of smoothed particle hydrodynamics data. *IEEE Transactions on Visualization and Computer Graphics*, 15:1243–, 11 2009. doi: 10.1109/TVCG.2009.173

[30] D. Sujudi and R. Haimes. Identification of swirling flow in 3-d vector fields. *AIAA Paper*, pp. 792–800, 1995. doi: 10.2514/6.1995-1715

[31] H. Theisel, J. Sahner, T. Weinkauf, H.-C. Hege, and H.-P. Seidel. Extraction of parallel vector surfaces in 3D time-dependent fields and application to vortex core line tracking. In *Proc. IEEE Visualization*, pp. 631–638, 2005. doi: 10.1109/VISUAL.2005.1532851

[32] H. Theisel and H.-P. Seidel. Feature flow fields. In *Proceedings of the Symposium on Data Visualisation 2003*, VISSYM '03, pp. 141–148. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2003.

[33] X. Tricoche, C. Garth, T. Bobach, G. Scheuermann, and M. Rütten. Accurate and efficient visualization of flow structures in a delta wing simulation. In *34th AIAA Fluid Dynamics Conference and Exhibit*, p. 2153, 2004. doi: 10.2514/6.2004-2153

[34] A. Van Gelder and A. Pang. Using pvsolve to analyze and locate positions of parallel vectors. *IEEE Transactions on Visualization and Computer Graphics*, 15(4):682–695, 2009. doi: 10.1109/TVCG.2009.11

[35] T. Weinkauf, J. Sahner, H. Theisel, and H.-C. Hege. Cores of swirling particle motion in unsteady flows. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1759–1766, 2007. doi: 10.1109/TVCG.2007.70545

[36] T. Weinkauf, H. Theisel, A. Van Gelder, and A. Pang. Stable feature flow fields. *IEEE Transactions on Visualization and Computer Graphics*, 17(6):770–780, June 2011. doi: 10.1109/TVCG.2010.93