Eurographics Conference on Visualization (EuroVis) 2026
N. Andrienko, M. Chen, and B. Wang
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 45 (2026), Number 3

Parallel Vectors Extraction using Bézier Clipping

Nico Daßler and Tobias Günther

Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

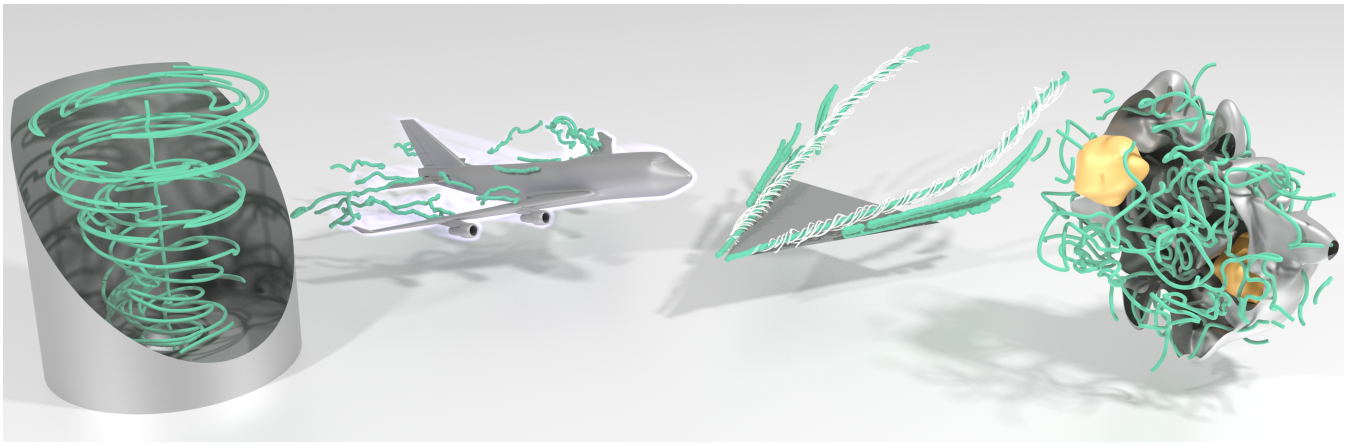


Figure 1: *We propose a novel parallel vectors (PV) extractor [PR99], which is based on Bézier clipping [SN90]–a root finding solver that is popular in computer-aided geometric design. Our method achieves better convergence than Bézier bisection and readily generalizes to polynomial higher-order interpolants. Our test scenes from left to right:* SWIRLING JET, BOEING, DELTA WING, *and* BORROMEAN.

Abstract

In this paper, we propose a novel local feature extraction algorithm for the parallel vectors (PV) operator. Our method is based on Bézier clipping, which is a bracketing-based root finding method that is commonly-used in computer-aided geometric design. Compared to Bézier bisection, our clipping method exhibits significantly faster convergence and reduces duplicate solutions. Compared to the general Newton-Raphson algorithm, the success of our approach does not depend on the quality of an initial guess. The Bézier-based formulation readily generalizes to polynomial higher-order interpolants, such as for tricubic interpolation. With this, we derive the acceleration in the Sujudi-Haimes vortex coreline criterion directly from the vector field interpolant. In addition, we describe the cross product of two polynomial vector fields in arbitrary polynomial degree, which can be used to approximate non-polynomial interpolants. Further, we examine the effect of Newton refinement on the proposed Bézier clipping method. We evaluate our Bézier clipping method thoroughly on a range of different data sets.

CCS Concepts

• ***Human-centered computing*** → ***Scientific visualization;*** • ***Computing methodologies*** → *Computer graphics;*

1. Introduction

In many disciplines of science and engineering, such as in meteorology, aircraft engineering, or electromagnetism, the state of dynamical systems is frequently described by means of scalar, vector, and tensor fields. To develop a deeper understanding of the underlying physical processes, it is necessary to determine the key ingredients that govern the evolution of such fields. A branch of scientific visualization focuses in particular on the extraction of so-called features, which are geometric structures that are often times defined implicitly. More than 25 years ago, Peikert and Roth [PR99] established a common mathematical foundation for the description of feature lines by showing that many of them can be described formally as solutions to the so-called *Parallel Vectors* (PV) problem. In the PV problem, feature lines are defined implicitly as locations at which two feature-specific vector fields are parallel, which is the case when their cross product is zero. This simple and ele-

gant description spurred decades of research on the reformulation of common features as PV problems [PR99, RP98, PS08, BSB*22], and on numerical extraction algorithms that robustly find solutions to the operator [SFBP09, WG20, GP21, DG24]. On discrete data, values and derivatives are estimated from carefully chosen interpolants. One possible choice are *polynomial interpolants*, such as Bézier-splines or the non-uniform rational B-splines in multivariate functional approximations [PNG*18]. We identified several open challenges with polynomial interpolants in the PV operator:

- **Faster Convergence.** Once the cross product of the PV problem is phrased in Bernstein-Bézier representation, a recursive bisection algorithm [ORT18a, GT19] is available. Unlike Newton-Raphson, the bisection algorithm is a *bracketing method* that finds solutions on a given domain rather than requiring a good initial guess. However, not only does bisection converge slowly, as it always only halves the domain, it also exhibits duplicate roots due to numerics. In this work, we tailor another bracketing-based method called Bézier clipping [SN90] to the PV problem, which increases both convergence and numerical stability. To this end, we investigate and compare two clipping strategies: component-wise clipping and projection-based clipping.
- **Newton Refinement for Bézier Clipping.** The iterative Newton-Raphson method exhibits quadratic convergence but it might miss solutions if the initial guess was insufficient. One option is to apply the linearly converging Bézier bisection first to narrow down the candidate domain [BRG21], however, this loses the advantage of quadratic convergence. Since Bézier clipping exhibits quadratic convergence on its own [Sch09], it is not clear if there is merit to incorporating a Newton refinement. We evaluate the effect of Newton refinement for our Bézier clipping method, showing that Newton refinement is not necessary.
- **Derived Acceleration.** The parallel vectors reformulation of the reduced velocity criterion [SH95] searches for locations at which the velocity and the acceleration are parallel. For polynomial interpolants, the acceleration can be derived in closed-form. For a C^1 continuous cubic interpolant, we derive the Bézier form of acceleration directly from the velocity tensor product.
- **Independent Interpolants.** If the vectors in the cross product cannot be derived directly in polynomial form, e.g., when one side includes eigenvectors, a possible fallback is to approximate the non-polynomial field with a polynomial interpolant. For this use case, we describe the cross product of two polynomial vector fields of arbitrary polynomial degree, which we evaluate for bilinear and bicubic interpolation on cell faces.

We evaluate our methods within a comprehensive benchmark on multiple vector fields, comprising a detailed analysis of the runtime and the algorithmic behavior (convergence, clipping rates, etc.). Fig. 1 gives examples of our method, in which we extracted parallel vectors feature curves, including a SWIRLING JET, a BOEING 747-400 aircraft at a 15° angle of attack, a DELTA WING, and the electromagnetic field of BORROMEAN rings (yellow).

2. Related Work

2.1. Root Finding Problems in Bézier Form

Bézier Basics. Originating from computer-aided geometric design [Far02], a Bézier curve $\mathbf{x}(t) : [t_0, t_1] \to \mathbb{R}^m$ with polynomial degree n is expressed as linear combination of its control points $\mathbf{b}_i$:

$$\mathbf{x}(t) = \sum_i^n B_i^n(t) \cdot \mathbf{b}_i, \qquad B_i^n(t) = \binom{n}{i} \cdot t^i \cdot (1-t)^{n-i}, \quad (1)$$

where $B_i^n(t)$ are the Bernstein basis functions with $i \in \{1, \dots, n\}$. Its generalization is called tensor product, here for two dimensions:

$$\mathbf{x}(u,v) = \sum_i^{n_1} \sum_j^{n_2} B_i^{n_1}(u) \cdot B_j^{n_2}(v) \cdot \mathbf{b}_{i,j}. \quad (2)$$

Any polynomial function can be expressed in Bézier form. Due to the linearity of the Bernstein basis functions, the conversion from any other linear basis, such as the monomial basis $\{1, t, \dots, t^n\}$, can be done through a linear transformation, which we elaborate on later in more detail. A key property of Bézier representations is that the convex hull of its control points forms a strict outer bound for the curve, which is useful when computing roots.

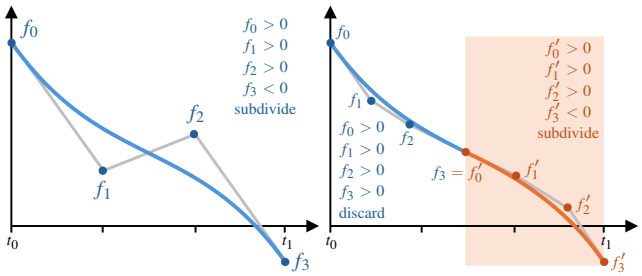
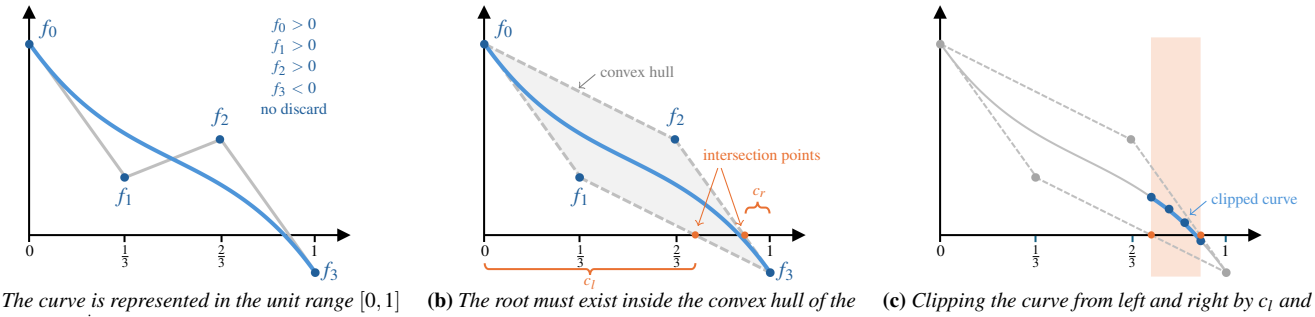


Figure 2: *Illustration of the bracketing-based Bézier bisection algorithm. Left: If all control point values are positive ($\forall i : f_i > 0$), or all are negative ($\forall i : f_i < 0$), then no roots can exist. Here, $f_0, f_1, f_2 > 0$ and $f_3 < 0$, which means that a root might exist. Right: After subdivision, the left half can be discarded ($\forall i : f_i > 0$), while the right half recursively continues subdivision. Note that only 50% of the domain can be removed with each recursion.*

Bézier Bisection. To convey the general idea, we consider the one-dimensional, uni-variate root finding problem $f(t) = 0$ in the range $[t_0, t_1]$, where $f(t) = \sum_i^n B_i^n(t) \cdot f_i$ is in Bézier form. The *Bézier bisection* method [RHD89, HLS93] utilizes that function $f(t)$ cannot contain roots in $[t_0, t_1]$ if all control points are strictly larger ($\forall i : f_i > 0$) or all are strictly smaller ($\forall i : f_i < 0$) than zero. If this is not the case, then the range $[t_0, t_1]$ may or may not contain roots. In this case, the curve is subdivided into two subcurves $f_1(t) : [t_0, \frac{t_0+t_1}{2}] \to \mathbb{R}$ and $f_2(t) : [\frac{t_0+t_1}{2}, t_1] \to \mathbb{R}$, and the sign check is performed recursively for each subcurve. An example is shown in Fig. 2. By the Weierstrass approximation theorem, the convex hull fits progressively tighter to the original function. The procedure is repeated recursively until the parameter range $[t'_0, t'_1]$ becomes sufficiently small, i.e., $t'_1 - t'_0 \leq \epsilon$. Since the method has linear convergence it requires a high number of recursions to reach high precision, which requires many Bézier curve subdivisions. Additionally, duplicates may arise due to floating point imprecision.

Bézier Clipping. On its recursive path towards a root, the bisection method removes exactly 50% of the parameter range with each recursion. The *Bézier clipping* method [SN90] is another bracketing method that instead tries to make even more progress with



(a) *The curve is represented in the unit range* $[0,1]$ *with points* $(\frac{i}{n}, f_i)$. *If* $\forall i : f_i > 0$ *or* $\forall i : f_i < 0$, *no root exists and recursion stops. Not the case here.*

(b) *The root must exist inside the convex hull of the control points. Intersecting the horizontal axis with the convex hull gives clip ratios* c_l, c_r.

(c) *Clipping the curve from left and right by* c_l *and* c_r *results in a shorter subcurve. The straighter the curve, the more gets clipped. Continue recursively.*

Figure 3: *Illustration of the Bézier clipping algorithm for solving the root finding problem of a uni-variate, one-dimensional function.*

each recursion. Fig. 3 gives an example. Instead of naively splitting the curve into two pieces, the clipping method defines the points $(\frac{i}{n}, f_i)$, i.e., the function values are linearly spaced in the $[0,1]$ range. The method then utilizes that the root must be contained within the convex hull of these points. The convex hull (although never explicitly calculated) is intersected with the horizontal axis, giving the relative clip ratios c_l and c_r. The Bézier-curve is then clipped from left and right to the subrange $[t_0', t_1']$, where $t_0' = t_0 + (t_1 - t_0) \cdot c_l$ and $t_1' = t_1 - (t_1 - t_0) \cdot c_r$. Again the Weierstrass approximation theorem guarantees the convergence to the exact function. However, this method gets stuck when there are multiple roots in the range $[t_0, t_1]$. If the procedure converges not fast enough ($c_l + c_r < 0.1$), then the curve is split once as in Bézier bisection and the process continues recursively. In practice, the clipping method often leads to clips of about 80%-99% percent of the parameter range, resulting in significantly faster convergence than pure Bézier bisection. Formally, the convergence rate of Bézier clipping was proven to be quadratic [Sch09]. In contrast to the Bézier bisection method, the clipping method is numerically more robust and produces less duplicate solutions.

Applications in CAGD. The Bézier clipping method has first been developed for the robust intersection of 2D Bézier curves [SN90]. The other major application domain of Bézier clipping is the calculation of ray intersections with tensor product surfaces. For (rational) Bézier and B-spline surfaces, the parameter ranges $u_{\min}, u_{\max}$ and $v_{\min}, v_{\max}$ are refined until a tolerance condition is satisfied [NSK90]. Thereby, distances are measured in the normal plane of the view ray, measured along two directions $\mathbf{c}_1$ and $\mathbf{c}_2$, such that these directions are approximately aligned with the surface tangents in u- and v-direction. Problems and strategies for selecting the next parameter direction (u or v) have been discussed by Campagna et al. [CSS97], who also proposed the use of Chebyshev boxes as proxies around Bézier patches. Efremov et al. [EHS05] addressed further view-dependent and numerical problems. For Bézier triangles [RDG01] the clipping is performed in barycentric coordinates (r,s,t) instead, i.e., $r_{\min}, r_{\max}$, $s_{\min}, s_{\max}$, and $t_{\min}, t_{\max}$.

Multivariate Function Approximation. Peterka et al. [PNG*18] proposed approximating spatial field data using non-uniform rational B-splines (NURBS). Apart from enabling smooth evaluation of values and derivatives, the method can compress data by forming coarser approximations under a desired error tolerance. By now, the MFA construction has been accelerated [GPMN19, MLGP24], was made adaptive [PLGM23], and was applied in-situ [PNG*22], in direct volume rendering [SLYP23, SLYP24], and in topology [MLP*24, MLG*25]. The B-splines in MFA can be converted to Bézier tensor products by linearly transforming the control points from de Boor to Bézier, and by lifting into homogeneous coordinates to handle the rational weights [SN90]. Ma et al. [MLP*24] used the iterative Newton-Raphson method to locate critical points. In contrast, we develop a bracketing-based Bézier clipping method that does not require adequate initial guesses.

2.2. Parallel Vectors Operator

Peikert and Roth [PR99] introduced the Parallel Vectors (PV) operator, which searches for all locations at which two continuous vector fields $\mathbf{v}(\mathbf{x})$ and $\mathbf{w}(\mathbf{x})$ are parallel:

$$\mathbf{v}(\mathbf{x}) \parallel \mathbf{w}(\mathbf{x}) \quad \Leftrightarrow \quad \mathbf{v}(\mathbf{x}) \times \mathbf{w}(\mathbf{x}) = \mathbf{0}. \tag{3}$$

A number of feature curves have been defined in terms of the PV operator. We focus on vortex corelines in steady 3D flow using the reduced velocity criterion of Sujudi-Haimes [SH95], which we phrase in the eigenvector-free form [PR99]:

$$\mathbf{v}(\mathbf{x}) \parallel \underbrace{(\mathbf{J}(\mathbf{x}) \cdot \mathbf{v}(\mathbf{x}))}_{\mathbf{w}(\mathbf{x}) := \mathbf{a}(\mathbf{x})}. \tag{4}$$

With this, the field $\mathbf{w}(\mathbf{x})$ remains polynomial if $\mathbf{v}(\mathbf{x})$ was polynomial. Based on the eigenvalues of the Jacobian, the resulting feature line can be classified as vortex coreline (swirling motion) or bifurcation lines (saddle-like motion) [PC87, Rot00]. For time-dependent fluid flow, the vector field can be first transformed into a near-steady reference frame [GGT17, BRG19, HMTR19]. Apart from those features, the PV operator has been used among others to define bent corelines [RP98], ridge/valley lines [PS08], and jet streams [KHS*18, BSB*22].

Piecewise Linear Vector Fields. For two linear vector fields $\mathbf{v}(\mathbf{x})$ and $\mathbf{w}(\mathbf{x})$, Peikert and Roth [PR99] have shown that the search for

PV solutions on tetrahedral cell faces is done by solving an eigenanalysis. Later, Guo et al. [GP21], have shown that for two linear tetrahedral elements, an analytic solution can be found. For (higher-dimensional) piecewise-linear fields, roots have been searched on the faces of tetrahedra via Bézier bisection [ORT18a,GT19], which bisects Bézíer representations of the cross product components recursively to converge to the root. In this paper, we are not concerned with piecewise linear approximations of a given vector field, and instead aim for solutions to trilinear and tricubic fields.

Local Techniques. Peikert and Roth [PR99] introduced multiple local PV extractors. One option is to intersect isocontours of two cross product components, which is numerically challenging when there is a small angle between isocontours. Alternatively, one can solve for roots on the faces of each grid cell and connect the solutions afterwards. If the connectivity is ambiguous, grid cells are recursively subdivided. To find the roots, Peikert and Roth [PR99] used the iterative Newton-Raphson method per cell face. Starting from the cell center as initial guess $\mathbf{x}_c^{(k)}$, and a differentiable multivariate function $\mathbf{g}(\mathbf{x})$, the method iterates [Kel95, Chp. 5]:

$$\mathbf{x}_c^{(k+1)} = \mathbf{x}_c^{(k)} - \left[\nabla \mathbf{g}(\mathbf{x}_c^{(k)})\right]^{-1} \cdot \mathbf{g}(\mathbf{x}_c^{(k)}). \tag{5}$$

When $\nabla \mathbf{g}$ is not invertible, a pseudoinverse is used instead. Since the cross product has a 1-dimensional null space, a sectional Newton descent was performed by Schindler et al. [SFBP09] and later by Witschi et al. [WG20] in the context of an implicit PV renderer. To reach high accuracy, a higher-order marching cubes with adaptive mesh refinement [RCMG07] is needed, which has the same convergence rate as the bisection method, as it recursively refines the domain via bisections. For a polynomial function in Bézier form, the bisection method can be used [ORT18a, GT19, BRG21], which terminates the recursion as soon as any of the three cross product components cannot contain a root anymore.

Integration-based Techniques. Integration-based approaches construct a feature flow field [TS03] for tracing the PV solution as tangent curve. To increase the numerical accuracy, a predictor-corrector style approach [VGP09] and a stable FFF [WTVGP10] have been developed. The FFF needs seed points, for which Theisel et al. [TSW*05] proposed an octree-based refinement that assumed trilinear interpolation. In general non-linear fields, Pagot et al. [POS*11] used reduced affine arithmetic (a form of interval arithmetic) [KHK*09] to locate candidate points, refined those with Newton iterations, and then traced feature curves using feature flow fields. In this work, we develop a fully local method for polynomial interpolants, which does not require initial guesses.

Related Problems. While the classic PV operator always considers two vector fields, Gerrits et al. [GRT18] searched for approximate PV solutions to multiple vector fields, while Oster et al. [ORT18b] searched for parallel eigenvectors in tensor fields. Another generalization is the high-dimensional dependent vectors operator [HS19], which searches for the k-dimensional space, where n-dimensional vector fields are parallel. For the parallel vectors operator, this is $n = 3$ and $k = 1$. Their approach traverses the $(n-k)$-faces of the n-dimensional grid and performs Gauss-Newton iterations to find isolated solutions, which are afterwards connected. Recently, the parallel vectors problem has been phrased as variational minimizer [DG24], for which a predictor-corrector and a refinement-based solver have been developed.

3. Method

With this paper, we develop a novelbracketing-based parallel vectors (PV) [PR99] solver that is based on Bézier clipping [SN90]. Following local PV extraction methods on regular grids, we solve for the entry and exit points of PV lines on all cell faces, which are afterwards connected with adaptive subdivision to handle connectivity ambiguities. In Section 3.1, we first derive the cross product of two vector fields on a cell face in general Bézier form, which applies to vector fields of arbitrary polynomial degree (bilinear, biquadratic, bicubic, etc.). The higher-order polynomial representations enable sampling of higher-order derivatives and can be used to approximate non-polynomial interpolants. In Section 3.2, we develop a C^1 continuous cubic interpolant, which converts from raw grid points to Bézier control points of tensor product volumes. In Section 3.3, we derive the acceleration in tensor product form directly from the cubic interpolant of the velocity, which we use in the eigenvector-free formulation of the Sujudi-Haimes criterion [SH95]. Lastly, Section 3.4 describes our extension of the Bézier clipping algorithm to the parallel vectors problem, for which we propose a component-wise and a projection-based algorithm. A C++ reference implementation is available at [DG26].

3.1. Cross Products in Bézier Form

Bézier form. On a quad face with parameterization $(u,v) \in [0,1]^2$, the two vector fields $\mathbf{v}(u,v)$ and $\mathbf{w}(u,v)$ can be represented in Bernstein-Bézier basis as tensor products [Far02], cf. Eq. (2):

$$\mathbf{v}(u,v) = \sum_{i=0}^{n_\mathbf{v}} \sum_{j=0}^{m_\mathbf{v}} B_i^{n_\mathbf{v}}(u) \cdot B_j^{m_\mathbf{v}}(v) \cdot \mathbf{v}_{i,j}, \tag{6}$$

$$\mathbf{w}(u,v) = \sum_{i=0}^{n_\mathbf{w}} \sum_{j=0}^{m_\mathbf{w}} B_i^{n_\mathbf{w}}(u) \cdot B_j^{m_\mathbf{w}}(v) \cdot \mathbf{w}_{i,j}. \tag{7}$$

The field $\mathbf{v}(u,v)$ is represented with $(n_\mathbf{v}+1) \times (m_\mathbf{v}+1)$ control points $\mathbf{v}_{i,j}$ with $0 \le i \le n_\mathbf{v}$ and $0 \le j \le m_\mathbf{v}$. Likewise, the field $\mathbf{w}(u,v)$ is represented with $(n_\mathbf{w}+1) \times (m_\mathbf{w}+1)$ control points $\mathbf{w}_{i,j}$ with $0 \le i \le n_\mathbf{w}$ and $0 \le j \le m_\mathbf{w}$. That is, we explicitly allow for a varying degree in the fields $\mathbf{v}(u,v)$ and $\mathbf{w}(u,v)$. This is important, since one of the fields is often computed from derived differential properties that may have a different polynomial degree.

Cross product. The cross product $\mathbf{c}(u,v) = \mathbf{v}(u,v) \times \mathbf{w}(u,v)$, which is solved by the parallel vectors operator in Eq. (3), is likewise represented in Bezier form:

$$\mathbf{c}(u,v) = \sum_{i=0}^{n_\mathbf{c}} \sum_{j=0}^{m_\mathbf{c}} B_i^{n_\mathbf{c}}(u) \cdot B_j^{m_\mathbf{c}}(v) \cdot \mathbf{c}_{i,j}, \tag{8}$$

where $n_\mathbf{c} = n_\mathbf{v} + n_\mathbf{w}$ and $m_\mathbf{c} = m_\mathbf{v} + m_\mathbf{w}$. Thus, the degree of the cross product is the sum of the degrees of the individual fields. For example, if $\mathbf{v}(u,v)$ and $\mathbf{w}(u,v)$ are both bilinear, then the cross product becomes biquadratic. The control points $\mathbf{c}_{i,j}$ can be computed in

closed-form by convolution over the indexing space, which gives:

$$\mathbf{c}_{i,j} = \sum_{k\in\mathcal{K}_i}\sum_{l\in\mathcal{L}_j} \frac{\binom{n_v}{k}\binom{n_w}{i-k}}{\binom{n_c}{i}} \cdot \frac{\binom{m_v}{l}\binom{m_w}{j-l}}{\binom{m_c}{j}} \cdot \left(\mathbf{v}_{k,l} \times \mathbf{w}_{i-k,j-l}\right), \quad (9)$$

with $\mathcal{K}_i = \{k \in \mathbb{Z} \mid \max(0, i-n_w) \le k \le \min(i, n_v)\}$ and $\mathcal{L}_j = \{l \in \mathbb{Z} \mid \max(0, j-m_w) \le l \le \min(j, m_v)\}$. To accelerate the computation, we precompute a weight table in which the binomial coefficients are cached, as well. Further details and a pseudocode are provided in the supplemental material.

3.2. Cubic Interpolant in Bézier Form

When constructing a higher-order interpolant from grid points, one can choose between local and global spline interpolants. The global interpolants require solving a global linear system to match continuity constraints across all spline segments, which results in a Bézier representation as in Eq. (2). The local interpolants give up one continuity degree to avoid the need for a global linear system. In our work, we choose a Catmull-Rom spline, which is local.

Catmull-Rom splines. Next, we construct a cubic interpolant, starting with the 1D case. We use Catmull-Rom splines, which yield interpolating C^1 continous cubic curves $\mathbf{p}(t)$ that pass through two points $\mathbf{p}_1$ and $\mathbf{p}_2$ with given tangents $\mathbf{m}_1$ and $\mathbf{m}_2$,

$$\mathbf{p}(t) = h_{00}(t)\cdot\mathbf{p}_1 + h_{10}(t)\cdot\mathbf{m}_1 + h_{01}(t)\cdot\mathbf{p}_2 + h_{11}(t)\cdot\mathbf{m}_2, \quad (10)$$

with the Hermite basis functions

$$h_{00}(t) = 2t^3 - 3t^2 + 1, \qquad h_{01}(t) = t^3 - 2t^2 + t, \quad (11)$$

$$h_{10}(t) = -2t^3 + 3t^2, \qquad h_{11}(t) = t^3 - t^2, \quad (12)$$

and where the tangents are estimated from additional points, i.e., $\mathbf{m}_1 = \frac{\mathbf{p}_2-\mathbf{p}_0}{2}$, $\mathbf{m}_2 = \frac{\mathbf{p}_3-\mathbf{p}_1}{2}$. Thus, in summary four points $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ are needed to construct the interpolating cubic polynomial. Catmull-Rom splines are described in monomial basis $\mathbf{t}(t) = (1, t, t^2, t^3)^\mathrm{T}$ as follows:

$$\mathbf{p}(t) = \mathbf{t}(t)^\mathrm{T} \cdot \underbrace{\begin{pmatrix} 0 & 1 & 0 & 0 \\ -1/2 & 0 & 1/2 & 0 \\ 1 & -5/2 & 2 & -1/2 \\ -1/2 & 3/2 & -3/2 & 1/2 \end{pmatrix}}_{\mathbf{M}_p} \cdot \underbrace{\begin{pmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{pmatrix}}_{\mathbf{P}}. \quad (13)$$

To convert from the monomial basis to the Bernstein basis, we likewise use a linear transformation:

$$\mathbf{t}(t)^\mathrm{T} = \begin{pmatrix} B_0^3(t) \\ B_1^3(t) \\ B_2^3(t) \\ B_3^3(t) \end{pmatrix}^\mathrm{T} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1/3 & 0 & 0 \\ 1 & 2/3 & 1/3 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}}_{\mathbf{M}_b}. \quad (14)$$

Since each transformation is linear, we can express the interpolating Catmull-Rom spline immediately in Bernstein-Bézier form:

$$\mathbf{p}(t) = \sum_{i=0}^{3} B_i^3(t)\cdot\mathbf{b}_i, \quad (15)$$

with the control points $\mathbf{b}_i$ being computed via linear transformation:

$$\begin{pmatrix} \mathbf{b}_0 \\ \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & 1 & 0 & 0 \\ -1/6 & 1 & 1/6 & 0 \\ 0 & 1/6 & 1 & -1/6 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\mathbf{M}=\mathbf{M}_b\cdot\mathbf{M}_p} \cdot \begin{pmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{pmatrix}. \quad (16)$$

Regular grids. In the following, we utilize a C^1-continuous tricubic interpolant per cell. Each cell has $2\times2\times2$ corner values. Adding another voxel in each direction gives a $4\times4\times4$ block, which is needed to construct C^1 continuity. In the following, let $\mathbf{p}_{i,j,k}$ be the values on the $4\times4\times4$ grid with $i,j,k \in \{0,1,2,3\}$. The Bernstein-Bézier coefficients of the C^1-continuous tricubic interpolant are computed using trilinear tensor contraction:

$$\mathbf{b}_{i,j,k} = \sum_{a=0}^{3}\sum_{b=0}^{3}\sum_{c=0}^{3} \mathbf{M}_{i,a}\cdot\mathbf{M}_{j,b}\cdot\mathbf{M}_{k,c}\cdot\mathbf{p}_{a,b,c}, \quad (17)$$

where $\mathbf{M}_{i,a}$, $\mathbf{M}_{j,b}$, $\mathbf{M}_{k,c}$ access the entries of the 4×4 matrix $\mathbf{M} = \mathbf{M}_b\cdot\mathbf{M}_p$, which is shown in Eq. (16). The contraction is performed successively per dimension, for which a pseudocode is listed in the supplemental material. With this, we are able to describe tricubic interpolation of vector fields in Bézier form. Note that multivariate functional approximations [PNG*18] likewise construct polynomial interpolants, but in non-uniform rational B-spline form.

3.3. Derived Acceleration

The eigenvector-free vortex coreline definition for the reduced velocity criterion [SH95] calculates $\mathbf{v} \parallel \mathbf{a}$. In the following, we derive the acceleration directly in Bézier form from the velocity.

Given the velocity field $\mathbf{v}(u,v,w)$ as tensor product of degree $n_1\times n_2\times n_3$, with the control points being computed with Eq. (17):

$$\mathbf{v}(u,v,w) = \sum_{i=0}^{n_1}\sum_{j=0}^{n_2}\sum_{k=0}^{n_3} B_i^{n_1}(u)\cdot B_j^{n_2}(v)\cdot B_k^{n_3}(w)\cdot\mathbf{v}_{i,j,k}. \quad (18)$$

First, the Jacobian $\mathbf{J}(u,v,w) = (\mathbf{v}_x(u,v,w), \mathbf{v}_y(u,v,w), \mathbf{v}_z(u,v,w))$ is computed. Its first column vector is the velocity partial $\frac{\partial\mathbf{v}}{\partial x}$, which has degree $(n_1-1)\times n_2\times n_3$. Similarly, $\frac{\partial\mathbf{v}}{\partial y}$ has degree $n_1\times(n_2-1)\times n_3$ and $\frac{\partial\mathbf{v}}{\partial z}$ has degree $n_1\times n_2\times(n_3-1)$. Each partial is represented as tensor product:

$$\frac{\partial\mathbf{v}(u,v,w)}{\partial x} = \sum_{i=0}^{n_1-1}\sum_{j=0}^{n_2}\sum_{k=0}^{n_3} B_i^{n_1-1}(u)\cdot B_j^{n_2}(v)\cdot B_k^{n_3}(w)\cdot\partial_x\mathbf{v}_{i,j,k}, \quad (19)$$

$$\frac{\partial\mathbf{v}(u,v,w)}{\partial y} = \sum_{i=0}^{n_1}\sum_{j=0}^{n_2-1}\sum_{k=0}^{n_3} B_i^{n_1}(u)\cdot B_j^{n_2-1}(v)\cdot B_k^{n_3}(w)\cdot\partial_y\mathbf{v}_{i,j,k}, \quad (20)$$

$$\frac{\partial\mathbf{v}(u,v,w)}{\partial z} = \sum_{i=0}^{n_1}\sum_{j=0}^{n_2}\sum_{k=0}^{n_3-1} B_i^{n_1}(u)\cdot B_j^{n_2}(v)\cdot B_k^{n_3-1}(w)\cdot\partial_z\mathbf{v}_{i,j,k}, \quad (21)$$

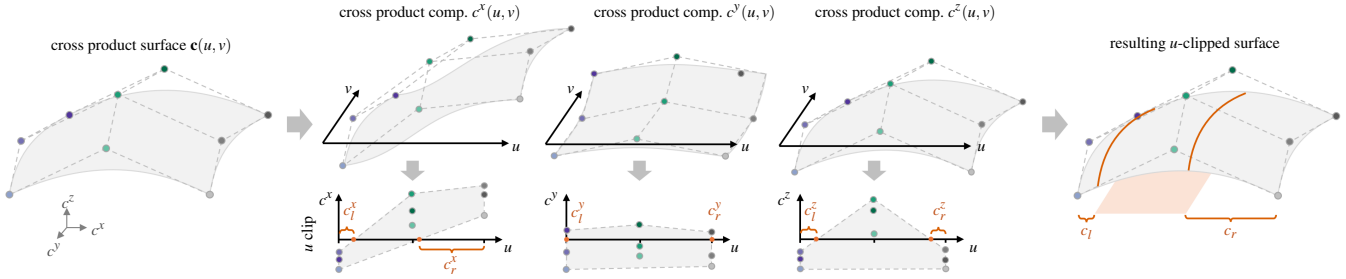


Figure 4: *Component-wise Bézier clipping of the cross product surface* $\mathbf{c}(u,v) = (c^x(u,v), c^y(u,v), c^z(u,v))^\mathrm{T}$ *(left). Each cross-product component can be seen as a height surface (middle top). To perform a u-clip, the rows of control points are inserted per component into a plot with the coordinates* $(\frac{i}{n_u}, c^x_{i,j})$, $(\frac{i}{n_u}, c^y_{i,j})$, *and* $(\frac{i}{n_u}, c^z_{i,j})$, *respectively (middle bottom). Intersecting the convex hull with the u-axis gives rise to a relative clip range per component, i.e.,* c^x_l, c^x_r *and* c^y_l, c^y_r *and* c^z_l, c^z_r. *Since a root must exist on all components, the surface* $\mathbf{c}(u,v)$ *is u-clipped with* $c_l = \max(\{c^x_l, c^y_l, c^z_l\})$ *and* $c_r = \min(\{c^x_r, c^y_r, c^z_r\})$ *(right). This can be done analogously for a v-clip.*

where the control points are computed by difference formulas:

$$\partial_x \mathbf{v}_{i,j,k} = n_1 \cdot (\mathbf{v}_{i+1,j,k} - \mathbf{v}_{i,j,k})/\Delta_x, \tag{22}$$

$$\partial_y \mathbf{v}_{i,j,k} = n_2 \cdot (\mathbf{v}_{i,j+1,k} - \mathbf{v}_{i,j,k})/\Delta_y, \tag{23}$$

$$\partial_z \mathbf{v}_{i,j,k} = n_3 \cdot (\mathbf{v}_{i,j,k+1} - \mathbf{v}_{i,j,k})/\Delta_z, \tag{24}$$

with $\Delta_x = \frac{\partial x}{\partial u}$, $\Delta_y = \frac{\partial y}{\partial v}$, $\Delta_z = \frac{\partial z}{\partial w}$ denoting the grid spacing. As visible in the summation bounds in Eqs. (19)–(21), each partial derivative has a different degree. This complicates the computation of the acceleration, since Bernstein polynomials would have to be evaluated for different degrees. To unify the treatment and to simplify the computation, we perform a degree elevation to the degree $n_1 \times n_2 \times n_3$, here at the example for $\partial_x \mathbf{v}_{i,j,k}$:

$$\partial_x \mathbf{v}'_{i,j,k} = \begin{cases} \frac{n_1 - i}{n_1} \cdot \partial_x \mathbf{v}_{i,j,k}, & i = 0 \\ \frac{i}{n_1} \cdot \partial_x \mathbf{v}_{i-1,j,k}, & i = n_1 \\ \frac{n_1 - i}{n_1} \partial_x \mathbf{v}_{i,j,k} + \frac{i}{n_1} \cdot \partial_x \mathbf{v}_{i-1,j,k}, & \text{else.} \end{cases} \tag{25}$$

With this, the acceleration field $\mathbf{a}(u,v,w) = \mathbf{J}(u,v,w) \cdot \mathbf{v}(u,v,w)$ can likewise conveniently be expressed in tensor product form with the degree $2n_1 \times 2n_2 \times 2n_3$:

$$\mathbf{a}(u,v,w) = \sum_{i=0}^{2n_1} \sum_{j=0}^{2n_2} \sum_{k=0}^{2n_3} B_i^{2n_1}(u) \cdot B_j^{2n_2}(v) \cdot B_k^{2n_3}(w) \cdot \mathbf{a}_{i,j,k}, \tag{26}$$

with the control points being matrix-vector products of the elevated Jacobian control points and the velocity control points:

$$\mathbf{a}_{i,j,k} = \sum_{\substack{i'+i''=i \\ j'+j''=j \\ k'+k''=k}} \frac{\binom{n_1}{i'}\binom{n_1}{i''}}{\binom{2n_1}{i}} \frac{\binom{n_2}{j'}\binom{n_2}{j''}}{\binom{2n_2}{j}} \frac{\binom{n_3}{k'}\binom{n_3}{k''}}{\binom{2n_3}{k}} \cdot \left(\mathbf{J}'_{i',j',k'} \cdot \mathbf{v}_{i'',j'',k''}\right), \tag{27}$$

with $\mathbf{J}'_{i',j',k'} = (\partial_x \mathbf{v}'_{i',j',k'}, \partial_y \mathbf{v}'_{i',j',k'}, \partial_z \mathbf{v}'_{i',j',k'})$ being the elevated Jacobian from Eq. (25). Thereby, the indices i', j', k' loop the control points of $\mathbf{J}'$ and the indices i'', j'', k'' loop the control points of $\mathbf{v}$. Again, weight tables are precomputed to accelerate the computation, which is shown in form of pseudocode in the supplemental material. For the purpose of vortex coreline extraction according to the Sujudi-Haimes criterion, the faces of the velocity and acceleration voxel are chosen as $\mathbf{v}(u,v)$ and $\mathbf{w}(u,v)$ field, respectively, see Eqs. (6)–(7). Here, for example, the lower z cell face:

$$\mathbf{v}_{i,j} = \mathbf{v}_{i,j,0}, \qquad \mathbf{w}_{i,j} = \mathbf{a}_{i,j,0}, \tag{28}$$

with the degrees $n_\mathbf{v} = n_1$, $m_\mathbf{v} = n_2$, $n_\mathbf{w} = 2n_1$, and $m_\mathbf{w} = 2n_2$.

3.4. Bézier Clipping for Parallel Vectors

In the following, we describe two recursive Bézier clipping algorithms for the extraction of parallel vectors solution on cell faces, i.e., for the detection of solutions to $\mathbf{c}(u,v) := \mathbf{v}(u,v) \times \mathbf{w}(u,v) = (0,0,0)^\mathrm{T}$. Both clipping algorithms decide in each iteration whether to clip in u-direction or in v-direction. We discuss the decision process at the end of this section. To explain the clipping procedure, we will first assume that the clip is done in u-direction.

Component-wise Clipping. The Bézier bisection algorithm in Section 2.1 can be applied component-wise, when solving the parallel vectors problem [BRG21]. With each recursion, a subinterval is only continued if the existence of a root cannot be excluded on any of the three components. Similarly, the Bézier clipping algorithm can be executed component-wise, too. An illustration of the algorithm is shown in Fig. 4. The cross-product field $\mathbf{c}(u,v) = (c^x(u,v), c^y(u,v), c^z(u,v))^\mathrm{T}$ is tri-variate. Each of the three scalar components can be thought of as a height surface, as shown in Fig. 4 (middle top). We begin with the first component $c^x(u,v)$. Similar to Fig. 3, we could take the first row of its control points $c^x_{0,0}, \ldots, c^x_{n_u,0}$ and plot them over u as points $(\frac{i}{n_u}, c^x_{i,0})$. Intersecting the convex hull of those control points with the u-axis gives us the relative clip range (from left and right) for which the curve $c^x(u,v)_{|v=0}$ cannot contain a root. To obtain the clip range for any v in $c^x(u,v)$, we need to insert *all* control points $(\frac{i}{n_u}, c^x_{i,j})$ with $i \in \{0, n_u\}$ and $j \in \{0, n_v\}$ into the plot, see Fig. 4 (middle bottom). Computing the convex hull gives us the clip ranges c^x_l (from left) and c^x_r (from right). Doing this similarly for the second component $c^y(u,v)$ and third component $c^z(u,v)$ of $\mathbf{c}(u,v)$, we get the ranges c^y_l and c^y_r, as well as c^z_l and c^z_r. Since we require the roots to exist in all three components at the same time, we can clip the surface $\mathbf{c}(u,v)$ in all three components directly with, see Fig. 4 (right):

$$c_l = \max(\{c^x_l, c^y_l, c^z_l\}), \qquad c_r = \max(\{c^x_r, c^y_r, c^z_r\}). \tag{29}$$

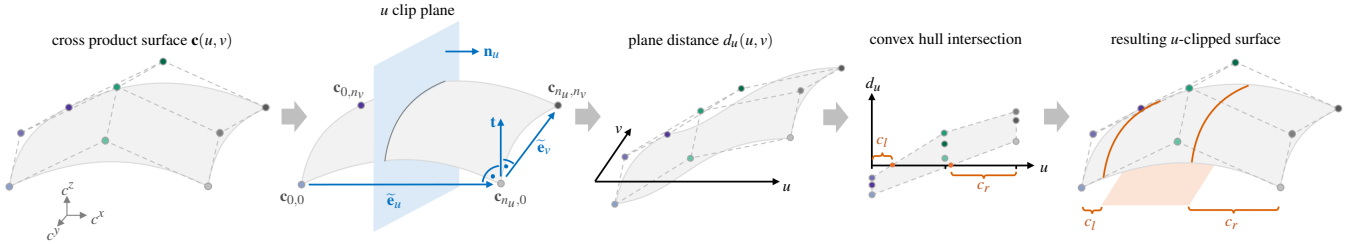


Figure 5: *Projection-based Bézier clipping computes for a given cross-product surface $\mathbf{c}(u,v)$ (first image), the direction $\mathbf{n}$ in which the clip cuts away from the surface, here for a u-clip (second image). Next, the distance $d_u(u,v)$ from the surface $\mathbf{c}(u,v)$ to the clip plane is computed in Bézier form (third image). Intersecting the convex hull of the points $(\frac{i}{n_u}, d^u_{i,j})$ with the u-axis gives the relative clipping ranges c_l, c_r (fourth image). Lastly, the Bézier surface is clipped, removing parts that cannot contain roots (fifth image). The process is similar for a v-clip.*

At iteration k, the Bézier surface is thereby u-clipped from left and right from the domain $[u_0^{(k)}, u_1^{(k)}]$ to the new range $[u_0^{(k+1)}, u_1^{(k+1)}]$:

$$u_0^{(k+1)} = u_0^{(k)} + \left(u_1^{(k)} - u_0^{(k)}\right) \cdot c_l, \tag{30}$$

$$u_1^{(k+1)} = u_1^{(k)} - \left(u_1^{(k)} - u_0^{(k)}\right) \cdot c_r. \tag{31}$$

The clipping can be done with the de Casteljau algorithm [Far02], or to speed it up, with the Volk-Schumaker algorithm [SV86]. Thus, in summary, we can calculate the clipping ranges component-wise, which requires three convex hull calculations. Each convex hull can be computed with a plane sweep at a runtime complexity of $\mathcal{O}(n_u \cdot n_v)$. Thus, the cost rises for higher-order polynomial vector fields.

Projection-based Clipping. To lower the number of clipping range computations, we transfer the ray intersection algorithm of Efremov et al. [EHS05] to the parallel vectors problem. The cross product surface $\mathbf{c}(u,v)$ has three components and can be pictured as a surface in 3D space parameterized by (u,v), see Fig. 5 (first image). The goal is to cut pieces from the surface in either u- or v-direction, until the resulting surface patch converges sufficiently far onto the point $(0,0,0)^{\mathrm{T}}$. The idea of the projection-based clipping is to ask how far a surface point $\mathbf{c}(u,v)$ is away from the root $(0,0,0)^{\mathrm{T}}$, measured along the direction in which the clipping is cutting away from the surface. This way, surface points that are clipped at the same time have equal distance, resulting in more compact convex hulls, and thereby larger clips. When clipping in u direction, then the clip cuts on average orthogonal to the average v-tangent:

$$\mathbf{e}_v = \frac{1}{|u_1 - u_0| \cdot |v_1 - v_0|} \int_{u_0}^{u_1} \int_{v_0}^{v_1} \frac{\partial}{\partial v} \mathbf{c}(u', v')_{|u=u', v=v'} \, \mathrm{d}u' \mathrm{d}v'. \tag{32}$$

Reversely, a clipping in v direction, cuts on average orthogonal to the average u-tangent:

$$\mathbf{e}_u = \frac{1}{|u_1 - u_0| \cdot |v_1 - v_0|} \int_{u_0}^{u_1} \int_{v_0}^{v_1} \frac{\partial}{\partial u} \mathbf{c}(u', v')_{|u=u', v=v'} \, \mathrm{d}u' \mathrm{d}v'. \tag{33}$$

Neglecting the scaling, the *direction* of these average tangents is approximated from the outer corners of the control polygon:

$$\widetilde{\mathbf{e}}_u = \mathbf{c}_{n_u,0} - \mathbf{c}_{0,0} + \mathbf{c}_{n_u,n_v} - \mathbf{c}_{0,n_v}, \qquad \frac{\widetilde{\mathbf{e}}_u}{\|\widetilde{\mathbf{e}}_u\|} \approx \frac{\mathbf{e}_u}{\|\mathbf{e}_u\|}, \tag{34}$$

$$\widetilde{\mathbf{e}}_v = \mathbf{c}_{0,n_v} - \mathbf{c}_{0,0} + \mathbf{c}_{n_u,n_v} - \mathbf{c}_{n_u,0}, \qquad \frac{\widetilde{\mathbf{e}}_v}{\|\widetilde{\mathbf{e}}_v\|} \approx \frac{\mathbf{e}_v}{\|\mathbf{e}_v\|}. \tag{35}$$

To measure the distance from the origin $(0,0,0)^{\mathrm{T}}$ orthogonal to the direction $\widetilde{\mathbf{e}}_v$ (for a u-clip) or $\widetilde{\mathbf{e}}_u$ (for a v-clip), we construct two implicit plane equations. First, we construct a vector $\mathbf{t} = \widetilde{\mathbf{e}}_u \times \widetilde{\mathbf{e}}_v$, which approximates the average normal of the cross product surface $\mathbf{c}(u,v)$. Then, we construct the clipping plane normals to be orthogonal to both $\mathbf{t}$ and the respective surface tangent $\widetilde{\mathbf{e}}_u$ or $\widetilde{\mathbf{e}}_v$:

$$\mathbf{n}_u = \frac{\widetilde{\mathbf{e}}_v \times \mathbf{t}}{\|\widetilde{\mathbf{e}}_v \times \mathbf{t}\|}, \qquad \mathbf{n}_v = \frac{\widetilde{\mathbf{e}}_u \times \mathbf{t}}{\|\widetilde{\mathbf{e}}_u \times \mathbf{t}\|}. \tag{36}$$

Fig. 5 (second image) illustrates the construction for a u-clip. The distance from a point $\mathbf{p}$ to the implicit planes is calculated simply by inserting into the Hesse normal form of the planes:

$$d_u(\mathbf{p}) = \mathbf{n}_u^{\mathrm{T}}\mathbf{p} + \mathbf{o}, \qquad d_v(\mathbf{p}) = \mathbf{n}_v^{\mathrm{T}}\mathbf{p} + \mathbf{o}, \tag{37}$$

with $\mathbf{o} = (0,0,0)^{\mathrm{T}}$ since we measure the distance to the origin, i.e., the root. To measure how far any surface point $\mathbf{c}(u,v)$ is away from the implicit planes, we insert $\mathbf{c}(u,v)$:

$$d_u(u,v) := d_u(\mathbf{c}(u,v)) = \sum_{i=0}^{n_u} \sum_{j=0}^{n_v} B_i^{n_u}(u) \cdot B_j^{n_v}(v) \cdot \underbrace{\mathbf{n}_u^{\mathrm{T}}\mathbf{c}_{i,j}}_{d^u_{i,j}} \tag{38}$$

$$d_v(u,v) := d_v(\mathbf{c}(u,v)) = \sum_{i=0}^{n_u} \sum_{j=0}^{n_v} B_i^{n_u}(u) \cdot B_j^{n_v}(v) \cdot \underbrace{\mathbf{n}_v^{\mathrm{T}}\mathbf{c}_{i,j}}_{d^v_{i,j}} \tag{39}$$

As we can see from Eqs. (38)–(39), the distance functions $d_u(u,v)$ (for a u-clip) and $d_v(u,v)$ (for a v-clip) can be expressed directly in Bézier form, cf. Fig. 5 (third image) for an illustration of the distance surface $d_u(u,v)$ for a u-clip. From here, the Bézier clipping proceeds as in the component-wise algorithm. Roots are guaranteed to not exist if $\forall i,j : d^u_{i,j} > 0$ or $\forall i,j : d^u_{i,j} < 0$ (and likewise for v). For a u-clip, we compute the convex hull of the points $(\frac{i}{n_u}, d^u_{i,j})$, which directly gives the relative clipping ranges c_l (from left) and c_r (from right), see Fig. 5 (fourth image). Lastly, the cross product surface $\mathbf{c}(u,v)$ is clipped to remove parts that certainly do not contain roots, see Fig. 5 (fifth image). The clipping in v-direction is analogous.

Clip Decision and Termination. During the iterative clipping process, we keep track of the parameter ranges $[u_0^{(k)}, u_1^{(k)}]$ and $[v_0^{(k)}, v_1^{(k)}]$, cf. Eqs. (30)–(31). The next clipping direction is decided

based on the progress in the two dimensions, favoring the direction in which more progress can be made:

$$\begin{cases} u\text{-clip}: & \frac{u_1^{(k+1)}-u_0^{(k+1)}}{u_1^{(k)}-u_0^{(k)}} \leq \frac{v_1^{(k+1)}-v_0^{(k+1)}}{v_1^{(k)}-v_0^{(k)}} \\ v\text{-clip}: & \text{else} \end{cases}. \tag{40}$$

The clipping process proceeds recursively until two termination conditions are reached, i.e., the clipped range is small enough:

$$u_1^{(k)} - u_0^{(k)} < \tau_{\text{range}} \quad \wedge \quad v_1^{(k)} - v_0^{(k)} < \tau_{\text{range}}, \tag{41}$$

and the cross product norm is small enough at the mid-point, which is eventually reported as the root:

$$\left\| \mathbf{c}\left(\frac{1}{2}(u_0^{(k)} + u_1^{(k)}), \frac{1}{2}(v_0^{(k)} + v_1^{(k)}) \right) \right\| < \tau_{\text{residual}}. \tag{42}$$

If one range is finished, e.g., $u_1^{(k)} - u_0^{(k)} < \tau_{\text{range}}$, then clipping continues only on the other range. In case of multiple roots, the progress slows down as with regular Bézier clipping [SN90]. If less than 10% of the range is clipped, the surface is split in the middle at $\frac{u_0^{(k)}+u_1^{(k)}}{2}$ for a u-clip, and at $\frac{v_0^{(k)}+v_1^{(k)}}{2}$ for a v-clip. This ensures that the convergence rate remains high throughout the process and it ensures that all roots are found. In our tests, we observed that the projection-based clipping strategy led to larger clips and thereby a speedup of about 2.7× to 3.9×, see later in Table 1, which is why we followed this approach in the remainder of this work.

4. Result

We evaluated our parallel vectors solvers on vortex/bifurcation line extraction tasks in the DELTA WING, BORROMEAN, SWIRLING-JET, and BOEING data set. We chose the numerical precision thresholds τ_{residual} and τ_{range} such that the bisection and the clipping method both find the *same* solutions, which enables a fair comparison of their runtime. As with any other PV solver, voxels can be discarded early based on the type of feature, for example to exclude voxels with insufficient vorticity ($\omega(\mathbf{x}) \leq \omega_{\text{min}}$). We used the following filters in our data sets: $\omega_{\text{min}} = 100$ (DELTA WING), $\omega_{\text{min}} = 0.05$ (BORROMEAN), and $\omega_{\text{min}} = 10$ (BOEING).

4.1. Solver Performance

We evaluated the parallel vectors extraction algorithms on a workstation with an AMD Ryzen 9 9950X CPU with 4.30GHz. For all measurements, we report the mean and standard deviation of 10 repeated measurements. We list the runtime and several metrics for the independent bilinear interpolant in Table 1. Additional measurements for the independent bicubic interpolant and for the bicubic interpolant with derived acceleration can be found in the supplemental material. Since Bézier bisection and Bézier clipping both compute the Bézier representation, cf. Sections 3.1–3.3, we separately list the *Solve Runtime*, which measures the runtime of the solver alone (Section 3.4). Fig. 6 shows a significant reduction in solve runtime to about 9.1× when using projection-based Bézier clipping compared to Bézier bisection, which is evident for both the bilinear and bicubic interpolant. In general, the bicubic interpolant is about 8.2× slower than the bilinear interpolant, for both bisection and clipping, see our supplemental material. This is because the bilinear interpolant solves for the roots in biquadratic cross product surfaces (3×3 control points), while the bicubic interpolant solves for the roots on surfaces with degree 6 in u and v direction (7×7 control points). Fortunately, the solve runtime is output sensitive, i.e., it depends on the number of solutions (*# Solutions*) and on the average recursion depth (*⌀ Sol. Depth*) at which those solutions were found. The performance advantage of the clipping method is directly reflected in the serial runtime. However, for datasets with sparse solutions (DELTA WING and NINE CL) the total runtime is dominated by the computation of the Bézier representations. Running the extraction in parallel further improved performance. However, the unbalanced workload across voxels negatively affects the thread utilization. We leave the optimization of the workload scheduler to future work. For our purposes, we used OpenMP's dynamic scheduler. We observed that the iterative Newton-Raphson method, as used by Peikert and Roth [PR99], is consistently slower than our clipping approaches and misses 0.3–9% of all PV point solutions, as it depends on the initial guess. The additional material contains visual comparisons and experiments with the bicubic interpolants, where Newton-Raphson misses up to 15% of solutions.

4.2. Solver Convergence

Two primary factors determine the faster solve runtime of the Bézier clipping approach. First, clipping produces less duplicates than bisection. With both methods, a root is discarded if it is too close to an existing one. Duplicate solutions may arise whenever the surface is split close to a root. Both subsurfaces then converge to approximately the same location on their shared boundary. The clipping solver only subdivides surfaces when the clip progress is too small (see Section 3.4), which might still give rise to duplicate solutions. However, this situation is rare and occurred only in the SWIRLING JET. In contrast, the bisection method subdivides the surface in every iteration, making duplicates more likely. With the clipping approach, the total number of duplicates remains significantly lower compared to the bisection approach. The second reason for the difference in solve runtime is the faster convergence of the clipping procedure. Fig. 7 shows the average recursion depths when a solution is found and the average clip sizes in the SWIRLING JET. The recursion depth is decreased significantly while the average clip size jumps from a fixed 50% for bisection to around 90% for clipping. This pattern is the same across all other datasets, see columns ⌀ Sol. Depth and ⌀ Clip Size in Table 1. With bicubic interpolation, an increase in recursion depth and a reduced clip size are observable, which in shown in the supplemental material. This is most likely attributed to the increased number of control points which makes finding an ideal plane more difficult. However, the differences in clip size and recursion depth stay large when comparing clipping and bisection, which is reflected in the reduced solve runtime for clipping.

4.3. Newton Refinement (Hybrid Solvers)

The Newton-Rapshon iterations in Eq. (5) exhibit quadratic convergence, but might miss solutions when given a poor initial guess. In contrast, Bézier bisection has only linear convergence, but since it is a bracketing method, the discovery of all roots within a

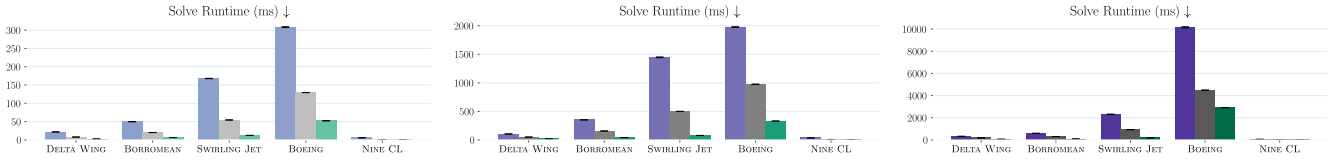


Figure 6: *Runtime comparison of Bézier-based approach. The solve runtime is measured alone because the computation of the Bézier representation is common to all methods. From left to right all possible datasets for a specific interpolant (bilinear, bicubic, derived acceleration) are compared showing the bisection solver in purple (●●●), the component-wise clipping solver in gray (●●●) and the projection-based clipping solver in green (●●●). On average we observe a speedup of* $3.1\times$ *of the component-wise clipping solver over the bisection solver and a speedup of* $2.9\times$ *of the projection-based clipping solver over the component-wise clipping solver.*

Dataset	Algorithm	Runtime (Serial)	Runtime (Parallel)	Solve Runtime	# Duplicates	# Solutions	⌀ Sol. Depth	⌀ Clip Size
Delta Wing	Bisection	89.85 ± 0.18	21.14 ± 1.78	21.79 ± 0.07	781	2276	66.33	0.500
	Clipping (C)	76.05 ± 0.38	8.40 ± 1.31	7.84 ± 0.05	85	2276	21.84	0.751
	Clipping (P)	74.07 ± 0.16	8.67 ± 1.04	2.66 ± 0.01	0	2276	7.46	0.925
	Newton-Raphson	74.40 ± 0.17	8.06 ± 1.39	12.23 ± 0.01	0	2213	—	—
	Piecewise Linear	554.82 ± 0.51	28.44 ± 1.26	—	0	2836	—	—
Borromean	Bisection	224.18 ± 0.47	35.87 ± 1.09	49.93 ± 0.06	5200	6971	36.97	0.500
	Clipping (C)	193.62 ± 0.18	13.71 ± 1.09	20.19 ± 0.08	633	6971	16.17	0.686
	Clipping (P)	184.27 ± 0.19	12.19 ± 1.26	6.53 ± 0.01	0	6971	5.71	0.959
	Newton-Raphson	296.06 ± 0.41	13.63 ± 0.69	38.03 ± 0.03	0	6948	—	—
	Piecewise Linear	7549.42 ± 2.25	333.11 ± 1.31	—	0	15507	—	—
Swirling Jet	Bisection	960.89 ± 3.06	139.67 ± 0.27	167.81 ± 0.49	12388	11769	52.17	0.500
	Clipping (C)	844.96 ± 0.19	57.30 ± 0.94	54.52 ± 0.23	2150	11769	24.71	0.632
	Clipping (P)	819.34 ± 0.14	53.83 ± 0.16	12.36 ± 0.06	55	11769	6.02	0.968
	Newton-Raphson	1305.83 ± 0.39	58.54 ± 0.77	108.82 ± 0.07	0	11728	—	—
	Piecewise Linear	34258.94 ± 25.02	1537.24 ± 7.70	—	0	18537	—	—
Boeing	Bisection	505.25 ± 2.44	135.44 ± 2.79	308.19 ± 1.14	21813	25752	49.79	0.500
	Clipping (C)	330.09 ± 1.86	43.95 ± 1.87	129.43 ± 0.24	2561	25752	24.42	0.650
	Clipping (P)	267.39 ± 5.24	30.27 ± 1.63	52.46 ± 0.50	0	25752	9.72	0.756
	Newton-Raphson	463.59 ± 1.51	34.22 ± 1.44	287.01 ± 0.34	0	22964	—	—
	Piecewise Linear	1432.61 ± 2.27	75.87 ± 1.71	—	0	51506	—	—
Nine CL	Bisection	358.00 ± 0.14	26.72 ± 1.04	5.87 ± 0.03	875	1625	42.69	0.500
	Clipping (C)	350.57 ± 0.15	24.27 ± 1.58	0.66 ± 0.00	0	1625	3.23	0.997
	Clipping (P)	358.30 ± 0.23	22.58 ± 0.37	0.88 ± 0.01	0	1625	4.15	0.996
	Newton-Raphson	586.37 ± 0.48	24.60 ± 0.87	19.67 ± 0.03	0	1250	—	—
	Piecewise Linear	6324.08 ± 4.63	293.25 ± 2.15	—	0	1500	—	—

Table 1: *Metrics recorded for independent bilinear interpolation to compare the performance on different datasets using the bisection method, our component-wise clipping (C), our projection-based clipping (P), and Newton-Raphson. For reference, the piecewise linear interpolant is listed, as well. All runtimes are measured and averaged for 10 runs, listing mean and standard deviation in milliseconds.*

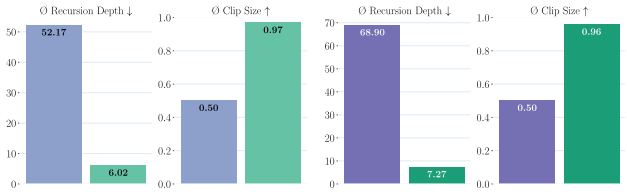


Figure 7: *Comparison of bilinear bisection (●), bicubic bisection (●), bilinear projection-based clipping (●), and bicubic projection-based clipping (●) in the* Swirling Jet.

given range is guaranteed. Combining the two in a *hybrid* approach [BRG21], starting with bisection and switching to Newton once it can be assumed that there is only one solution nearby, leads to improved convergence over bisection and less missed solutions than with Newton only [PR99]. However, this approach requires careful tuning of the parameter that controls when bisection is replaced by Newton iterations. In this section, we analyze the effect of Newton iterations on our clipping solver by collecting metrics for different values of this parameter (Newton start size) while fixing the residual termination parameter τ_{residual}, cf. Eq. (42). The implementation of this hybrid approach is straightforward. As soon as the clipping or bisection solver reaches a certain range τ_{range}, cf. Eq. (41), Newton iterations are performed until a certain error residual is reached or a maximum number of iterations is exceeded. The Jacobian necessary for the Newton iterations can be directly derived from the Bézier representation of the cross product, cf. Eqs. (19)–(21).

Fig. 8 examines the effect of decreasing the *Newton Start Size* on the Delta Wing. At the top, the solve runtime in milliseconds and the number of solutions are shown. The bottom row shows the av-

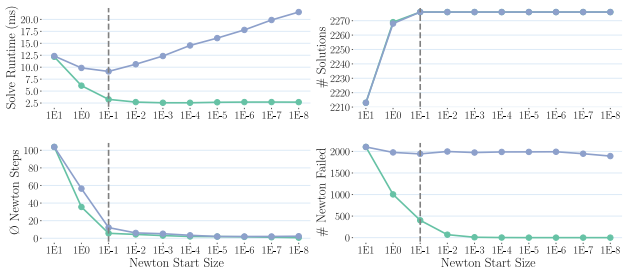


Figure 8: *Hybrid bisection (●) and hybrid clipping (●) in the* DELTA WING. *The convergence of Bézier clipping is not improved by using Newton steps, which is in contrast to the bisection solver. However, hybrid bisection requires tuning to not miss solutions. The completeness of solutions is shown by a dashed gray line.*

erage number of Newton steps and the number of times the Newton iterations did not converge to a root. The dashed line indicates the point when both solvers find the complete set of solutions. Using Newton only does not give all solutions, since the initial guesses are not close enough. Further, it runs slower than the hybrid approach, due to the large number of failed Newton refinements and a larger number of Newton steps. When using bisection first, the initial guesses for the Newton refinement improve. The best runtime performance was reached when all solutions were found. Lowering the *Newton Start Size* further increases the number of bisection steps, which are overall more expensive than Newton iterations. Meanwhile the average number of Newton steps decreases, while the number of failed Newton attempts stays high. This is due to the large number of duplicates that the bisection solver produces. Clipping quickly reduces the number of failed Newton attempts while the number of required Newton steps is approximately the same as for the bisection solver, which is expected behavior for the same start size. For the solve runtime, we observed no optimal point at which Newton should be started, showing that Bézier clipping does not converge significantly slower than the Newton approach. In fact it shows that Newton iterations, if not carefully tuned, harm the performance of the clipping solver. Thus, we recommend a hybrid approach only for the bisection solver, which requires appropriate tuning of the *Newton Start Size*. Our clipping approach does not benefit from Newton refinement.

4.4. Interpolants

Qualitative Comparison. Fig. 9 compares the independent bilinear, the independent bicubic, and the bicubic interpolant with derived acceleration in the BORROMEAN dataset. All lines were computed using projection-based clipping. For this dataset, we observed a smoother feature curve for the bicubic interpolant (left image) compared to the bilinear interpolant. As a result, the roots are more consistent and lead to smoother feature lines. In the right image, the bicubic interpolant with derived acceleration is compared against the bilinear interpolant. In some cases, the bi-cubic interpolant leads to smoother feature lines (top), while in other cases (left) the feature lines do not improve compared to the bilinear interpolant. The other datasets exhibit a similar behavior. This suggests that the right choice of interpolant depends on the data.

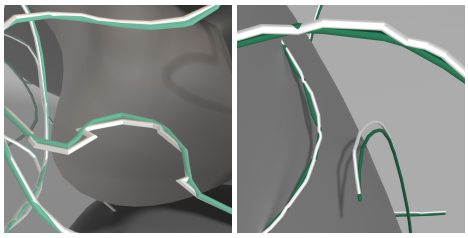


Figure 9: *Left: For some lines in the* BORROMEAN *dataset, we observed smoother results when using the bicubic interpolant (●) than when using bilinear interpolation (○). Right: With derived acceleration (●) some lines become smoother than with independent bilinear interpolation (○), while this may also fail for other lines.*

Choice of Interpolant. A higher-order interpolant, such as the bicubic interpolant, makes a difference when it was already present in the data, or when higher-order derivatives are needed. If the data has been discretized so finely that a bilinear interpolant is sufficient, then a bicubic interpolant will give very similar results. To demonstrate this, the supplemental material contains a comparison of different sampling resolutions on the analytic NINECL dataset.

Non-Polynomial Interpolant. If a data set has a non-polynomial interpolant, then tensor products can only approximate them to a desired error tolerance, which is a limitation we share with MFA [PNG*18]. An exact representation of a higher-order interpolant is only possible when the higher-order interpolant uses polynomial basis functions and when it is globally smooth such that no piece-wise behavior appears in the partial derivatives.

5. Conclusion

The parallel vectors operator [PR99] is a well-established mathematical framework for defining numerous feature curves in scalar and vector fields. In this paper, we introduced a novel bracketing-based parallel vectors solver, which solves the root finding problem on a parameter interval using Bézier clipping [SN90]. Compared to Bézier bisection, the clipping algorithm has faster convergence and less duplicate solutions. We derived the acceleration field for C^1 continuous tricubic interpolants in Bézier form, which we used in the Sujudi-Haimes vortex coreline criterion [SH95].

For the future, we would like to further accelerate the extraction by parallelizing computations on the GPU, which will require dedicated scheduling strategies, since the thread workload is non-uniformly distributed across the grid. Generalizations to parallel eigenvectors [ORT18b], approximate parallel vectors [GRT18], and dependent vectors [HS19] would be interesting future avenues. Since MFA [PNG*18] can be represented with Bézier tensor products after appropriate lifting, it would be interesting to apply Bézier clipping to MFA data, e.g., for critical point extraction.

Acknowledgements. We thank Markus Rütten for providing the DELTA WING, Simon Candelaresi for the BORROMEAN [CB11], Moritz Sieber for the SWIRLING JET [OSN*11], and Xingdi Zhang for the LBM simulation of the BOEING 747 [ZAT*26]. This work was supported by DFG grant no. GU 945/7-1.

References

[BRG19] BAEZA ROJO I., GÜNTHER T.: Vector field topology of time-dependent flows in a steady reference frame. *IEEE Transactions on Visualization and Computer Graphics 26*, 1 (2019), 280–290. doi:10.1109/TVCG.2019.2934375. 3

[BRG21] BAEZA ROJO I., GÜNTHER T.: Coreline criteria for inertial particle motion. In *Topological Methods in Data Analysis and Visualization IV*. Springer, 2021, pp. 133–157. doi:10.1007/978-3-030-83500-2_8. 2, 4, 6, 9

[BSB*22] BÖSIGER L., SPRENGER M., BOETTCHER M., JOOS H., GÜNTHER T.: Integration-based extraction and visualization of jet stream cores. *Geoscientific Model Development 15*, 3 (2022), 1079–1096. doi:10.5194/gmd-15-1079-2022. 2, 3

[CB11] CANDELARESI S., BRANDENBURG A.: Decay of helical and nonhelical magnetic knots. *Phys. Rev. E 84* (2011), 016406. doi:10.1103/PhysRevE.84.016406. 10

[CSS97] CAMPAGNA S., SLUSALLEK P., SEIDEL H.-P.: Ray tracing of spline surfaces: Bézier clipping, Chebyshev boxing, and bounding volume hierarchy–a critical comparison with new results. *The Visual Computer 13*, 6 (1997), 265–282. doi:10.1007/s003710050103. 3

[DG24] DASSLER N., GÜNTHER T.: Variational feature extraction in scientific visualization. *ACM Trans. Graph. (Proc. SIGGRAPH) 43*, 4 (July 2024). doi:10.1145/3658219. 2, 4

[DG26] DASSLER N., GÜNTHER T.: Reference implementation for parallel vectors extraction using Bézier clipping. https://github.com/fau-vc/pvBezier, 2026. Accessed: 2026-03-04. 4

[EHS05] EFREMOV A., HAVRAN V., SEIDEL H.-P.: Robust and numerically stable Bézier clipping method for ray tracing NURBS surfaces. In *Proceedings of the 21st Spring Conference on Computer Graphics* (New York, NY, USA, 2005), SCCG '05, Association for Computing Machinery, p. 127–135. doi:10.1145/1090122.1090144. 3, 7

[Far02] FARIN G.: *Curves and surfaces for CAGD: a practical guide*. Elsevier Science, Amsterdam, Netherlands, 2002. doi:10.1016/B978-1-55860-737-8.X5000-5. 2, 4, 7

[GGT17] GÜNTHER T., GROSS M., THEISEL H.: Generic objective vortices for flow visualization. *ACM Transactions on Graphics (Proc. SIGGRAPH) 36*, 4 (2017), 1–11. doi:10.1145/3072959.3073684. 3

[GP21] GUO H., PETERKA T.: Exact analytical parallel vectors. In *2021 IEEE Visualization Conference (VIS)* (New Orleans, LA, USA, 2021), IEEE, pp. 101–105. doi:10.1109/VIS49827.2021.9623310. 2, 4

[GPMN19] GRINDEANU I., PETERKA T., MAHADEVAN V. S., NASHED Y. S. G.: Scalable, high-order continuity across block boundaries of functional approximations computed in parallel. In *IEEE Conference on Cluster Computing (CLUSTER)* (2019), pp. 1–9. doi:10.1109/CLUSTER.2019.8891018. 3

[GRT18] GERRITS T., RÖSSL C., THEISEL H.: An approximate parallel vectors operator for multiple vector fields. *Computer Graphics Forum 37*, 3 (2018), 315–326. doi:10.1111/cgf.13422. 4, 10

[GT19] GÜNTHER T., THEISEL H.: Objective vortex corelines of finite-sized objects in fluid flows. *IEEE Transactions on Visualization and Computer Graphics (Proc. IEEE SciVis) 25*, 1 (Jan 2019). doi:10.1109/TVCG.2018.2864828. 2, 4

[HLS93] HOSCHEK J., LASSER D., SCHUMAKER L. L.: *Fundamentals of computer aided geometric design*. A. K. Peters, Ltd., USA, 1993. 2

[HMTR19] HADWIGER M., MLEJNEK M., THEUSSL T., RAUTEK P.: Time-dependent flow seen through approximate observer Killing fields. *IEEE Transactions on Visualization and Computer Graphics 25*, 1 (Jan 2019), 1257–1266. doi:10.1109/TVCG.2018.2864839. 3

[HS19] HOFMANN L., SADLO F.: The dependent vectors operator. *Computer Graphics Forum 38*, 3 (2019), 261–272. doi:10.1111/cgf.13687. 4, 10

[Kel95] KELLEY C. T.: Iterative methods for linear and nonlinear equations. *Society for Industrial and Applied Mathematics* (1995). doi:10.1137/1.9781611970944. 4

[KHK*09] KNOLL A., HIJAZI Y., KENSLER A., SCHOTT M., HANSEN C., HAGEN H.: Fast ray tracing of arbitrary implicit surfaces with interval and affine arithmetic. *Computer Graphics Forum 28*, 1 (2009), 26–40. doi:10.1111/j.1467-8659.2008.01189.x. 4

[KHS*18] KERN M., HEWSON T., SADLO F., WESTERMANN R., RAUTENHAUS M.: Robust detection and visualization of jet-stream core lines in atmospheric flow. *IEEE Transactions on Visualization and Computer Graphics 24*, 1 (2018), 893–902. doi:10.1109/TVCG.2017.2743989. 3

[MLG*25] MA G., LENZ D., GUO H., PETERKA T., WANG B.: Extracting complex topology from multivariate functional approximation: Contours, jacobi sets, and ridge-valley graphs. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)* (2025), pp. 54–64. doi:10.1109/LDAV68558.2025.00010. 3

[MLGP24] MAHADEVAN V., LENZ D., GRINDEANU I., PETERKA T.: Accelerating multivariate functional approximation computation with domain decomposition techniques. *Journal of Computational Science 78* (2024), 102268. doi:10.1016/j.jocs.2024.102268. 3

[MLP*24] MA G., LENZ D., PETERKA T., GUO H., WANG B.: Critical point extraction from multivariate functional approximation. In *2024 IEEE Topological Data Analysis and Visualization (TopoInVis)* (2024), pp. 12–22. doi:10.1109/TopoInVis64104.2024.00006. 3

[NSK90] NISHITA T., SEDERBERG T. W., KAKIMOTO M.: Ray tracing trimmed rational surface patches. *SIGGRAPH Comput. Graph. 24*, 4 (Sept. 1990), 337–345. doi:10.1145/97880.97916. 3

[ORT18a] OSTER T., RÖSSL C., THEISEL H.: Core lines in 3D second-order tensor fields. *Computer Graphics Forum 37*, 3 (2018), 327–337. doi:10.1111/cgf.13423. 2, 4

[ORT18b] OSTER T., RÖSSL C., THEISEL H.: The parallel eigenvectors operator. In *Vision, Modeling and Visualization* (Eindhoven, The Netherlands, 2018), The Eurographics Association, pp. 39–46. doi:10.2312/vmv.20181251. 4, 10

[OSN*11] OBERLEITHNER K., SIEBER M., NAYERI C. N., PASCHEREIT C. O., PETZ C., HEGE H.-C., NOACK B. R., WYGNANSKI I.: Three-dimensional coherent structures in a swirling jet undergoing vortex breakdown: stability analysis and empirical mode construction. *Journal of Fluid Mechanics 679* (2011), 383–414. doi:10.1017/jfm.2011.141. 10

[PC87] PERRY A. E., CHONG M. S.: A description of eddying motions and flow patterns using critical-point concepts. *Annual Review of Fluid Mechanics 19*, 1 (1987), 125–155. doi:10.1146/annurev.fl.19.010187.001013. 3

[PLGM23] PETERKA T., LENZ D., GRINDEANU I., MAHADEVAN V. S.: Towards adaptive refinement for multivariate functional approximation of scientific data. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)* (2023), pp. 32–41. doi:10.1109/LDAV60332.2023.00011. 3

[PNG*18] PETERKA T., NASHED Y. S. G., GRINDEANU I., MAHADEVAN V. S., YEH R., TRICOCHE X.: Foundations of multivariate functional approximation for scientific data. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)* (2018), pp. 61–71. doi:10.1109/LDAV.2018.8739195. 2, 3, 5, 10

[PNG*22] PETERKA T., NASHED Y., GRINDEANU I., MAHADEVAN V., YEH R., LENZ D.: Multivariate functional approximation of scientific data. In *In Situ Visualization for Computational Science* (Cham, 2022), Springer International Publishing, pp. 375–397. doi:10.1007/978-3-030-81627-8_17. 3

[POS*11] PAGOT C., OSMARI D., SADLO F., WEISKOPF D., ERTL T., COMBA J.: Efficient parallel vectors feature extraction from higher-order data. *Computer Graphics Forum 30*, 3 (2011), 751–760. doi:10.1111/j.1467-8659.2011.01924.x. 4

[PR99] PEIKERT R., ROTH M.: The "Parallel Vectors" operator – a vector field visualization primitive. In *Proc. IEEE Visualization* (San Francisco, CA, USA, 1999), IEEE, pp. 263–270. doi:10.1109/VISUAL.1999.809896. 1, 2, 3, 4, 8, 9, 10

[PS08] PEIKERT R., SADLO F.: Height ridge computation and filtering for visualization. In *2008 IEEE Pacific Visualization Symposium* (Kyoto, japan, 2008), IEEE, IEEE, pp. 119–126. doi:10.1109/PACIFICVIS.2008.4475467. 2, 3

[RCMG07] REMACLE J.-F., CHEVAUGEON N., MARCHANDISE É., GEUZAINE C.: Efficient visualization of high-order finite elements. *International Journal for Numerical Methods in Engineering 69*, 4 (2007), 750–771. doi:10.1002/nme.1787. 4

[RDG01] ROTH S. H. M., DIEZI P., GROSS M. H.: Ray tracing triangular Bézier patches. *Computer Graphics Forum 20*, 3 (2001), 422–432. doi:10.1111/1467-8659.00535. 3

[RHD89] ROCKWOOD A., HEATON K., DAVIS T.: Real-time rendering of trimmed surfaces. *SIGGRAPH Comput. Graph. 23*, 3 (July 1989), 107–116. doi:10.1145/74334.74344. 2

[Rot00] ROTH M.: *Automatic extraction of vortex core lines and other line-type features for scientific visualization*, vol. 9. ETH Zurich, Zurich, Switzerland, 2000. doi:10.3929/ethz-a-004016407. 3

[RP98] ROTH M., PEIKERT R.: A higher-order method for finding vortex core lines. In *Proc. IEEE Visualization* (Research Triangle Park, NC, USA, 1998), IEEE, pp. 143–150. doi:10.1109/VISUAL.1998.745296. 2, 3

[Sch09] SCHULZ C.: Bézier clipping is quadratically convergent. *Computer Aided Geometric Design 26*, 1 (2009), 61–74. doi:10.1016/j.cagd.2007.12.006. 2, 3

[SFBP09] SCHINDLER B., FUCHS R., BIDDISCOMBE J., PEIKERT R.: Predictor-corrector schemes for visualization ofsmoothed particle hydrodynamics data. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1243–1250. doi:10.1109/TVCG.2009.173. 2, 4

[SH95] SUJUDI D., HAIMES R.: *Identification of Swirling Flow in 3D Vector Fields*. Tech. rep., Departement of Aeronautics and Astronautics, MIT, 1995. AIAA Paper 95-1715. doi:10.2514/6.1995-1715. 2, 3, 4, 5, 10

[SLYP23] SUN J., LENZ D., YU H., PETERKA T.: Scalable volume visualization for big scientific data modeled by functional approximation. In *IEEE International Conference on Big Data (BigData)* (2023), pp. 905–914. doi:10.1109/BigData59044.2023.10386434. 3

[SLYP24] SUN J., LENZ D., YU H., PETERKA T.: MFA-DVR: direct volume rendering of MFA models. *Journal of Visualization 27*, 1 (2024), 109–126. doi:10.1007/s12650-023-00946-y. 3

[SN90] SEDERBERG T., NISHITA T.: Curve intersection using Bézier clipping. *Computer-Aided Design 22*, 9 (1990), 538–549. doi:10.1016/0010-4485(90)90039-F. 1, 2, 3, 4, 8, 10

[SV86] SCHUMAKER L. L., VOLK W.: Efficient evaluation of multivariate polynomials. *Computer Aided Geometric Design 3*, 2 (1986), 149–154. doi:10.1016/0167-8396(86)90018-X. 7

[TS03] THEISEL H., SEIDEL H.-P.: Feature flow fields. In *Proc. Symposium on Data Visualisation* (Grenoble, France, 2003), The Eurographics Association, pp. 141–148. doi:10.2312/VisSym/VisSym03/141-148. 4

[TSW*05] THEISEL H., SAHNER J., WEINKAUF T., HEGE H.-C., SEIDEL H.-P.: Extraction of parallel vector surfaces in 3D time-dependent fields and application to vortex core line tracking. In *Proc. IEEE Visualization* (Minneapolis, MN, USA, 2005), IEEE, pp. 631–638. doi:10.1109/VISUAL.2005.1532851. 4

[VGP09] VAN GELDER A., PANG A.: Using PVsolve to analyze and locate positions of parallel vectors. *IEEE Transactions on Visualization and Computer Graphics 15*, 4 (2009), 682–695. doi:10.1109/TVCG.2009.11. 4

[WG20] WITSCHI R., GÜNTHER T.: Implicit ray casting of the parallel vectors operator. In *IEEE Visualization Short Papers* (Utah, US, 2020), IEEE, pp. 31–35. doi:10.1109/VIS47514.2020.00013. 2, 4

[WTVGP10] WEINKAUF T., THEISEL H., VAN GELDER A., PANG A.: Stable feature flow fields. *IEEE Transactions on Visualization and Computer Graphics 17*, 6 (2010), 770–780. doi:10.1109/TVCG.2010.93. 4

[ZAT*26] ZHANG X., AGEELI A., THEUSSL T., HADWIGER M., RAUTEK P.: Exploring 3D unsteady flow using 6D observer space interactions. *IEEE Transactions on Visualization and Computer Graphics 32*, 1 (2026), 517–526. doi:10.1109/TVCG.2025.3642506. 10