Implicit Ray Casting of the Parallel Vectors Operator Additional Material

Ramon Witschi and Tobias Günther

1 Derivation Sectional Newton Descent

The third component of the cross product $\mathbf{f}$ vanishes to zero, if the first two components are already zero. To see this, assume that the first two components of $\mathbf{f}(\mathbf{x})$ are 0 and $w_3 \neq 0$:

$$\mathbf{f}(\mathbf{x}) = \mathbf{v}(\mathbf{x}) \times \mathbf{w}(\mathbf{x}) = \begin{pmatrix} v_2 w_3 - v_3 w_2 \\ v_3 w_1 - v_1 w_3 \\ v_1 w_2 - v_2 w_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ v_1 w_2 - v_2 w_1 \end{pmatrix} \tag{1}$$

The only exception is the special case of $\mathbf{v}(\mathbf{x}) \neq \mathbf{0}$, $\mathbf{w}(\mathbf{x}) \neq \mathbf{0}$ and $v_3 = w_3 = 0$, which is caught by the solver not converging [2]. Solving for $v_1 = \frac{v_3 w_1}{w_3}$ and $v_2 = \frac{v_3 w_2}{w_3}$ and inserting into Eq. (1) yields:

$$\begin{pmatrix} 0 \\ 0 \\ \frac{v_3 w_1}{w_3} w_2 - \frac{v_3 w_2}{w_3} w_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{2}$$

We can therefore consider only two components of $\mathbf{f}$ and follow Schindler et al. [2] in choosing the components f_k, f_l that maximize

$$k,l = \underset{k,l \in \{0,1,2\}}{\arg\max} \; \|\nabla f_k(\mathbf{x}_i) \times \nabla f_l(\mathbf{x}_i)\|_2 \,, \quad \text{s.t. } k \neq l \tag{3}$$

We can find the roots of $\mathbf{f}$ by considering only two components of

$$\nabla \mathbf{f}(\mathbf{x}_i) \cdot \mathbf{d} = -\mathbf{f}(\mathbf{x}_i) \tag{4}$$

which gives rise to the linear system

$$(\nabla f_k(\mathbf{x}_i), \nabla f_l(\mathbf{x}_i))^{\mathrm{T}} \cdot \mathbf{d} = -\begin{pmatrix} f_k(\mathbf{x}_i) \\ f_l(\mathbf{x}_i) \end{pmatrix} \tag{5}$$

where $\mathbf{d}$ is a linear combination of the two selected gradients:

$$\mathbf{d} = \alpha \nabla f_k(\mathbf{x}_i) + \beta \nabla f_l(\mathbf{x}_i) = (\nabla f_k(\mathbf{x}_i), \nabla f_l(\mathbf{x}_i)) \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{6}$$

Inserting Eq. (6) into Eq. (5) and rearranging for the weights α, β:

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix} = -\mathbf{A}^{-1} \cdot \begin{pmatrix} f_k(\mathbf{x}_i) \\ f_l(\mathbf{x}_i) \end{pmatrix} \tag{7}$$

with $\mathbf{A} = (\nabla f_k(\mathbf{x}_i), \nabla f_l(\mathbf{x}_i))^{\mathrm{T}} (\nabla f_k(\mathbf{x}_i), \nabla f_l(\mathbf{x}_i))$. Inserting the weights α, β from Eq. (7) into Eq. (6) leads to the expression

$$\mathbf{d} = -\left(\nabla f_k(\mathbf{x}_i),\ \nabla f_l(\mathbf{x}_i)\right) \cdot \mathbf{A}^{-1} \cdot \begin{pmatrix} f_k(\mathbf{x}_i) \\ f_l(\mathbf{x}_i) \end{pmatrix} \tag{8}$$

Implementation Details. During the sectional Newton descent, we use trilinear hardware interpolation to sample the vector fields $\mathbf{v}$ and $\mathbf{w}$, when evaluating the current residual. The filter field $s(\mathbf{x})$ can be sampled with nearest-interpolation. Our Newton descent employs a trust-region, which means that each step has an upper bound of one voxel. Further, if the residual increases, the step is revoked and the step size is decreased for the next attempt. Since the filter field $s(\mathbf{x})$ takes us close to a solution, we use in practice at most 10 iterations for the sectional Newton descent. Later in Fig. 2, we vary the number of iterations, showing that 10 is an adequate choice. Note that for 0 iterations, the filter field $s(\mathbf{x})$ is visualized.

Algorithm 1: Test if a parallel vectors solution may be contained on the face of a cell. Indices are zero-based.

Data: $\mathbf{v}_{00}, \mathbf{v}_{01}, \mathbf{v}_{10}, \mathbf{v}_{11}, \mathbf{w}_{00}, \mathbf{w}_{01}, \mathbf{w}_{10}, \mathbf{w}_{11} \in \mathbb{R}^3$
Result: *TRUE* or *FALSE*

for (i, j) *in* $\{(1,2),(2,0),(0,1)\}$ **do**

$b_0 := \mathbf{v}_{00}[i] \cdot \mathbf{w}_{00}[j] - \mathbf{v}_{00}[j] \cdot \mathbf{w}_{00}[i]$;
$b_2 := \mathbf{v}_{01}[i] \cdot \mathbf{w}_{01}[j] - \mathbf{v}_{01}[j] \cdot \mathbf{w}_{01}[i]$;
$b_6 := \mathbf{v}_{10}[i] \cdot \mathbf{w}_{10}[j] - \mathbf{v}_{10}[j] \cdot \mathbf{w}_{10}[i]$;
$b_8 := \mathbf{v}_{11}[i] \cdot \mathbf{w}_{11}[j] - \mathbf{v}_{11}[j] \cdot \mathbf{w}_{11}[i]$;
$b_1 := \frac{1}{2}(\mathbf{v}_{00}[i] \cdot \mathbf{w}_{01}[j] + \mathbf{v}_{01}[i] \cdot \mathbf{w}_{00}[j] - \mathbf{v}_{00}[j] \cdot \mathbf{w}_{01}[i] - \mathbf{v}_{01}[j] \cdot \mathbf{w}_{00}[i])$;
$b_3 := \frac{1}{2}(\mathbf{v}_{00}[i] \cdot \mathbf{w}_{10}[j] + \mathbf{v}_{10}[i] \cdot \mathbf{w}_{00}[j] - \mathbf{v}_{00}[j] \cdot \mathbf{w}_{10}[i] - \mathbf{v}_{10}[j] \cdot \mathbf{w}_{00}[i])$;
$b_5 := \frac{1}{2}(\mathbf{v}_{01}[i] \cdot \mathbf{w}_{11}[j] + \mathbf{v}_{11}[i] \cdot \mathbf{w}_{01}[j] - \mathbf{v}_{01}[j] \cdot \mathbf{w}_{11}[i] - \mathbf{v}_{11}[j] \cdot \mathbf{w}_{01}[i])$;
$b_7 := \frac{1}{2}(\mathbf{v}_{10}[i] \cdot \mathbf{w}_{11}[j] + \mathbf{v}_{11}[i] \cdot \mathbf{w}_{10}[j] - \mathbf{v}_{10}[j] \cdot \mathbf{w}_{11}[i] - \mathbf{v}_{11}[j] \cdot \mathbf{w}_{10}[i])$;
$b_4 := \frac{1}{4}(\mathbf{v}_{00}[i] \cdot \mathbf{w}_{11}[j] + \mathbf{v}_{01}[i] \cdot \mathbf{w}_{10}[j] + \mathbf{v}_{10}[i] \cdot \mathbf{w}_{01}[j] + \mathbf{v}_{11}[i] \cdot \mathbf{w}_{00}[j] - \mathbf{v}_{00}[j] \cdot \mathbf{w}_{11}[i] - \mathbf{v}_{01}[j] \cdot \mathbf{w}_{10}[i] - \mathbf{v}_{10}[j] \cdot \mathbf{w}_{01}[i] - \mathbf{v}_{11}[j] \cdot \mathbf{w}_{00}[i])$;
If $(b_k > 0 : \forall k)$ **return** *FALSE* ;
If $(b_k < 0 : \forall k)$ **return** *FALSE* ;

return *TRUE*;

2 Pseudo Code

2.1 Bézier-based Test if PV Solution Exists

The search for a PV solution is a multi-variate root finding problem:

$$\mathbf{f}(x,y) = \mathbf{v}(x,y) \times \mathbf{w}(x,y) = \begin{pmatrix} f_1(x,y) \\ f_2(x,y) \\ f_3(x,y) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{9}$$

By converting the cross product components into Bernstein basis $B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$:

$$f_k(x,y) = \sum_{i=0}^{2} \sum_{j=0}^{2} B_i^n(x) \cdot B_j^n(y) \cdot \mathbf{b}_{i,j}^k \tag{10}$$

the potential existence of a PV solution on the face of a cell can be tested using the convex hull property, as listed in Alg. 1.

2.2 Ray Marching Loop

Alg. 2 gives a high-level view of the inner steps of the ray marching loop. At each sample location, we first lookup the filter field to do empty space skipping. If we are in or nearby a voxel with potential PV solution, we perform the sectional Newton descent. If the descent does not converge to a solution, we return to advance on the ray with the next step. If a solution is found, a ray intersection test with the tube proxy is performed. On success, the normal is estimated and Lambertian shading is performed. Transfer functions are stored in look-up tables.

Algorithm 2: PV Operator inner ray-marching loop

Data: current position $\mathbf{x}$ on ray, filter field $s(\mathbf{x})$
Result: *RBGA* or *NONE*

if $\Big(\texttt{NOT}\ s(\mathbf{x})\Big)$ **return** *NONE*;

```
// Use root x* as the center of the cylinder.
```
$\mathbf{x}^* := \texttt{sectional\_Newton\_descent}(\mathbf{x})$;

if $\Big(\mathbf{x}^* = \mathit{NONE}$ **or** $\texttt{NOT}\ s(\mathbf{x}^*)\Big)$ **return** *NONE*;

```
// Cylinder orientation, see [3].
```
$\mathbf{t} := \texttt{compute\_tangent}(\mathbf{x}^*)$

```
// x̃ is the ray-cylinder intersection point.
```
$\tilde{\mathbf{x}} := \texttt{ray\_cylinder\_intersection}(\mathbf{x}^*, \mathbf{t})$;

if $\Big(\tilde{\mathbf{x}} = \mathit{NONE}\Big)$ **return** *NONE*;

```
// Compute cylinder normal at intersection.
```
$\mathbf{n} := \texttt{compute\_normal}(\mathbf{x}^*, \mathbf{t}, \tilde{\mathbf{x}})$;
return $\texttt{RGBA\_shading}(\mathbf{n})$;

Algorithm 3: Sampling of $\nabla \mathbf{f}$ in trilinearly interpolated vector fields $\mathbf{v}(\mathbf{x})$ and $\mathbf{w}(\mathbf{x})$. $(x,y,z)^{\mathrm{T}} \in [0,1]^3$ is the relative position in the cell, and $\mathrm{lerp}(a,b,t) = (1-t)\cdot a + t\cdot b$.

Data: $x,y,z \in \mathbb{R}$, $\mathbf{v}_{000}$, $\mathbf{v}_{001}$, $\mathbf{v}_{010}$, $\mathbf{v}_{011}$, $\mathbf{v}_{100}$, $\mathbf{v}_{101}$, $\mathbf{v}_{110}$, $\mathbf{v}_{111}$, $\mathbf{w}_{000}$, $\mathbf{w}_{001}$, $\mathbf{w}_{010}$, $\mathbf{w}_{011}$, $\mathbf{w}_{100}$, $\mathbf{w}_{101}$, $\mathbf{w}_{110}$, $\mathbf{w}_{111} \in \mathbb{R}^3$
Result: $\nabla \mathbf{f} = (\mathbf{f}_x, \mathbf{f}_y, \mathbf{f}_z)$

$\mathbf{h}_1 := \mathrm{lerp}(\mathrm{lerp}(\mathbf{w}_{001}-\mathbf{w}_{000}, \mathbf{w}_{101}-\mathbf{w}_{100}, x),$
$\quad \mathrm{lerp}(\mathbf{w}_{011}-\mathbf{w}_{010}, \mathbf{w}_{111}-\mathbf{w}_{110}, x), y)$;
$\mathbf{h}_2 := \mathrm{lerp}(\mathrm{lerp}(\mathbf{v}_{001}-\mathbf{v}_{000}, \mathbf{v}_{101}-\mathbf{v}_{100}, x),$
$\quad \mathrm{lerp}(\mathbf{v}_{011}-\mathbf{v}_{010}, \mathbf{v}_{111}-\mathbf{v}_{110}, x), y)$;
$\mathbf{h}_3 := \mathrm{lerp}(\mathrm{lerp}(\mathbf{w}_{010}-\mathbf{w}_{000}, \mathbf{w}_{110}-\mathbf{w}_{100}, x),$
$\quad \mathrm{lerp}(\mathbf{w}_{011}-\mathbf{w}_{001}, \mathbf{w}_{111}-\mathbf{w}_{101}, x), z)$;
$\mathbf{h}_4 := \mathrm{lerp}(\mathrm{lerp}(\mathbf{v}_{010}-\mathbf{v}_{000}, \mathbf{v}_{110}-\mathbf{v}_{100}, x),$
$\quad \mathrm{lerp}(\mathbf{v}_{011}-\mathbf{v}_{001}, \mathbf{v}_{111}-\mathbf{v}_{101}, x), z)$;
$\mathbf{h}_5 := \mathrm{lerp}(\mathrm{lerp}(\mathbf{w}_{100}-\mathbf{w}_{000}, \mathbf{w}_{110}-\mathbf{w}_{010}, y),$
$\quad \mathrm{lerp}(\mathbf{w}_{101}-\mathbf{w}_{001}, \mathbf{w}_{111}-\mathbf{w}_{011}, y), z)$;
$\mathbf{h}_6 := \mathrm{lerp}(\mathrm{lerp}(\mathbf{v}_{100}-\mathbf{v}_{000}, \mathbf{v}_{110}-\mathbf{v}_{010}, y),$
$\quad \mathrm{lerp}(\mathbf{v}_{101}-\mathbf{v}_{001}, \mathbf{v}_{111}-\mathbf{v}_{011}, y), z)$;
$\mathbf{h}_7 := \mathrm{lerp}(\mathrm{lerp}(\mathrm{lerp}(\mathbf{v}_{000}, \mathbf{v}_{100}, x), \mathrm{lerp}(\mathbf{v}_{010}, \mathbf{v}_{110}, x), y),$
$\quad \mathrm{lerp}(\mathrm{lerp}(\mathbf{v}_{001}, \mathbf{v}_{101}, x), \mathrm{lerp}(\mathbf{v}_{011}, \mathbf{v}_{111}, x), y), z)$;
$\mathbf{h}_8 := \mathrm{lerp}(\mathrm{lerp}(\mathrm{lerp}(\mathbf{w}_{000}, \mathbf{w}_{100}, x), \mathrm{lerp}(\mathbf{w}_{010}, \mathbf{w}_{110}, x), y),$
$\quad \mathrm{lerp}(\mathrm{lerp}(\mathbf{w}_{001}, \mathbf{w}_{101}, x), \mathrm{lerp}(\mathbf{w}_{011}, \mathbf{w}_{111}, x), y), z)$
;
$\mathbf{f}_x := \mathbf{h}_6 \times \mathbf{h}_8 + \mathbf{h}_7 \times \mathbf{h}_5$;
$\mathbf{f}_y := \mathbf{h}_4 \times \mathbf{h}_8 + \mathbf{h}_7 \times \mathbf{h}_3$;
$\mathbf{f}_z := \mathbf{h}_2 \times \mathbf{h}_8 + \mathbf{h}_7 \times \mathbf{h}_1$;

2.3 Sampling the Cross-Product Jacobian

Solving for the direction $\mathbf{d}$ of the next step in Eq. (8) requires the gradients of the cross-product components, i.e., ∇f_1, ∇f_2, and ∇f_3. In Alg. 3, we provide the pseudo-code for the sampling of the cross-product Jacobian of two trilinearly interpolated vector fields. Note that the gradients of the cross-product components ∇f_1, ∇f_2 and ∇f_3 are found in the rows of the Jacobian of the cross-product $\nabla \mathbf{f}$:

$$\nabla \mathbf{f} = \begin{pmatrix} | & | & | \\ \mathbf{f}_x & \mathbf{f}_y & \mathbf{f}_z \\ | & | & | \end{pmatrix} = \begin{pmatrix} - & \nabla f_1^{\mathrm{T}} & - \\ - & \nabla f_2^{\mathrm{T}} & - \\ - & \nabla f_3^{\mathrm{T}} & - \end{pmatrix} \tag{11}$$

Sphere Approximation. Instead of approximating a line locally with cylinders, it is possible to approximate it as union of spheres, which avoids the estimation of the tangent direction $\mathbf{t}$. In our experiments, spheres and cylinders produced mostly the same results at the same frame rate. Differences become apparent when using a small radius: Lines spanned by oriented cylinders cover slightly more screen space and are therefore longer visible, see Fig. 3. The adequate choice of the ray marching step size has a greater effect on the perceived continuity than the choice between local sphere or cylinder approximation.

3 Parameter Study

Step Size. In Fig. 1, we varied the step size of the ray marcher. With sufficiently small step size, the lines are not skipped and aliasing is avoided. In practice, we used a half voxel size.

Descent Iterations. In our experiments, a 10 iterations were sufficient to obtain visually satisfying results in all experiments, since the descent starts close to the solution. There was no further visual improvement, when using more iterations. If the cross product residual is below a numerical epsilon, the descent terminates sooner.

Viewport Resolution. 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 Fig. 4, we studied how the performance scales for varying viewport resolutions. We tested resolutions of 500×500, 1000×1000, etc. At a very high resolution of 4000×4000, the performance can drop to about $1.5\,\mathit{sec}$ per frame. For moderate display resolutions, our hardware achieves interactive performance, which is suitable for parameter exploration of feature extraction algorithms.

4 Explicit Particle-based Extraction

The explicit extraction of PV solutions, however, is not fast enough for real-time parameter exploration, such as vortex criterion selection, application of filters, or suitable subtraction of ambient motion. Thus, we introduced an implicit PV renderer, which can serve as a preview tool. If explicit geometry is required, two options are available: particle-based extraction and grid-based extraction, which can be applied once afterwards. For example, Kindlmann et al. [1] applied their descent method to a large set of particles in order to extract a geometric representation. In Fig. 5, we did this similarly with our descent approach. In the Swirling Jet, a point cloud of 33,619 vertices is constructed in 0.6 seconds on the CPU, where each point has a residual cross product norm of under 10^{-8}. Further, we used a grid-based extraction to compute the explicit groundtruth solutions, using the Bezier-based subdivision algorithm described in Alg. 1. Note that this experiment does not include the extraction of connected lines from the particles.

5 Performance Benchmark Details

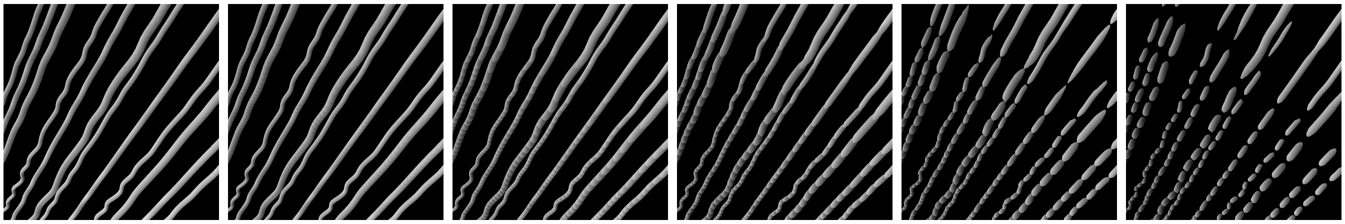
Please find in Fig. 6 the camera configurations used in Table 1 of the main paper for the benchmark results. In the supplementary video, we show a real-time capture of the Borromean Rings time series including the interactive volume rendering of the vorticity magnitude. There, we interactively change parameters with the visible frame rate to give an impression of the performance in interactive use.

References

[1] 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.

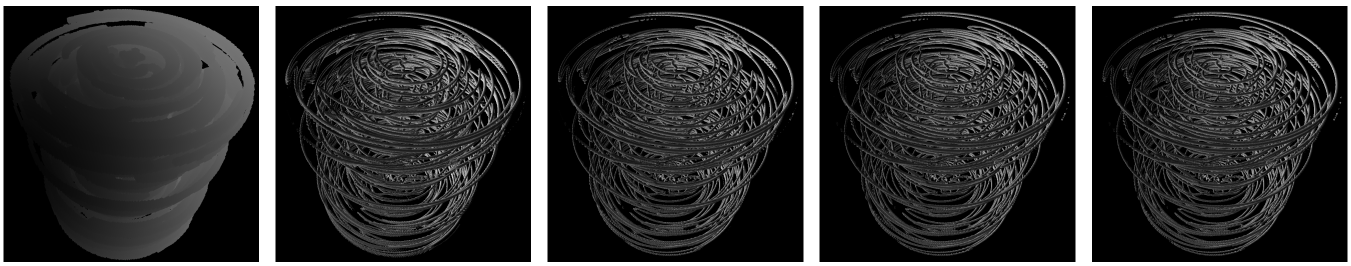
[2] B. Schindler, R. Fuchs, J. Biddiscombe, and R. Peikert. Predictor-corrector schemes for visualization of smoothed particle hydrodynamics data. *IEEE Transactions on Visualization and Computer Graphics*, 15:1243–, 11 2009. doi: 10.1109/TVCG.2009.173

[3] 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



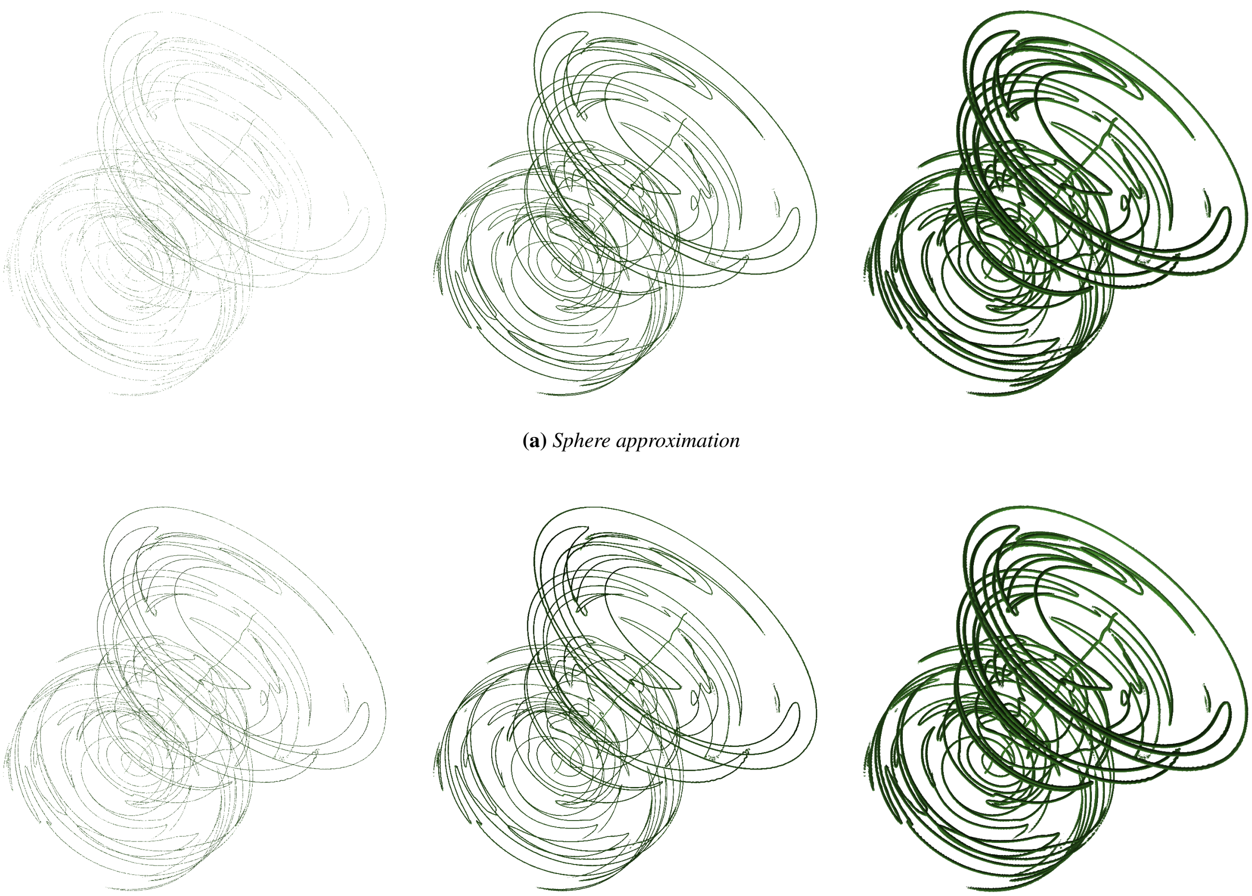
(a) *step size = 0.1 voxels* **(b)** *step size = 0.5 voxels* **(c)** *step size = 1.0 voxels* **(d)** *step size = 1.5 voxels* **(e)** *step size = 2.0 voxels* **(f)** *step size = 2.5 voxels*

Figure 1: *Comparison of varying ray marching step sizes, shown here for a close-up of the* CYLINDER *flow.*



0 iterations 1 iterations 2 iterations 5 iterations 10 iterations

Figure 2: *Results for varying number of sectional Newton descent iterations in the* SWIRLING JET *using the* $\mathbf{v} \parallel \mathbf{b}$ *criterion.*



(a) *Sphere approximation*

(b) *Cylinder approximation*

Figure 3: *Comparison of spheres and cylinders used to span the line geometry during the implicit ray casting with varying radii:* 0.1 *(left),* 0.3 *(middle) and* 1.0 *(right). Shown for the* SWIRLING JET *data set. Differences can be seen for small radii in the left-most column.*

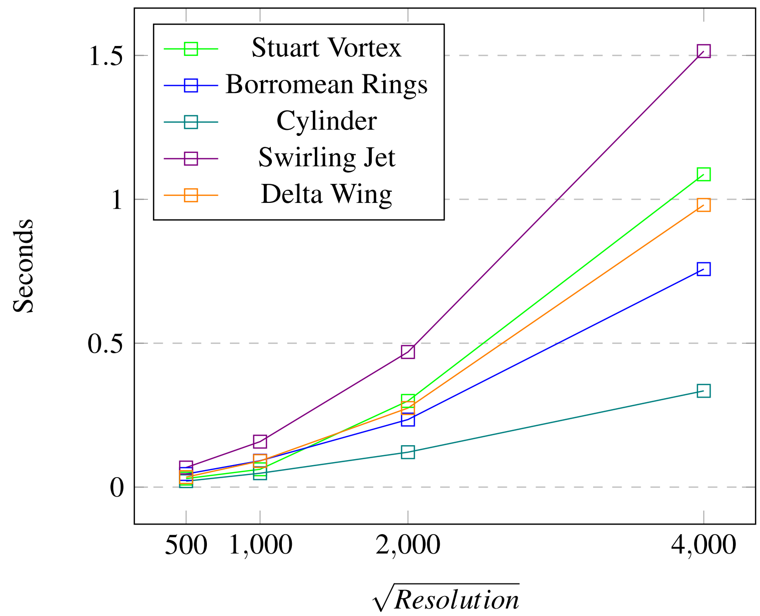
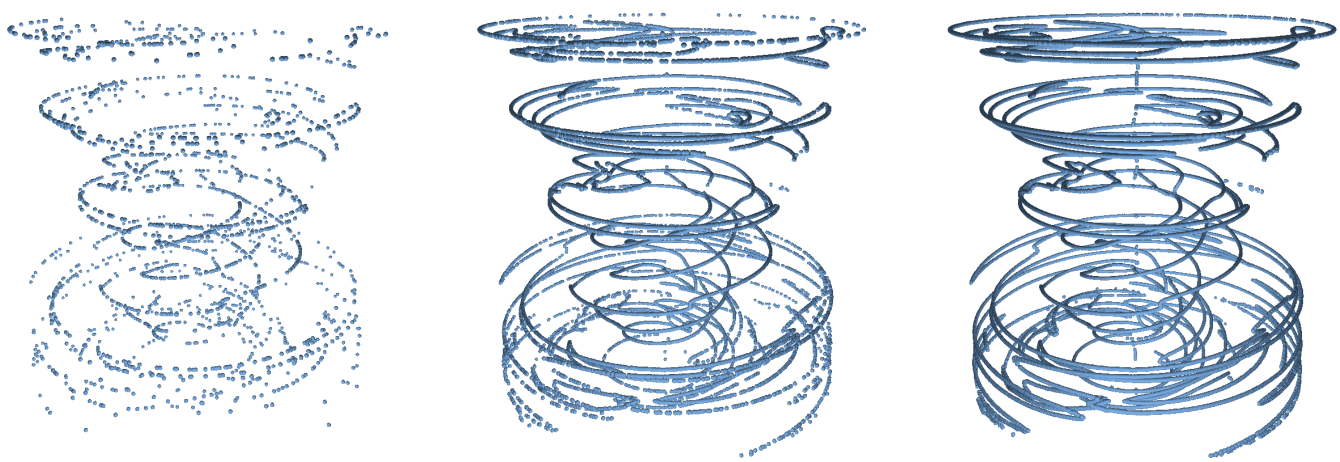
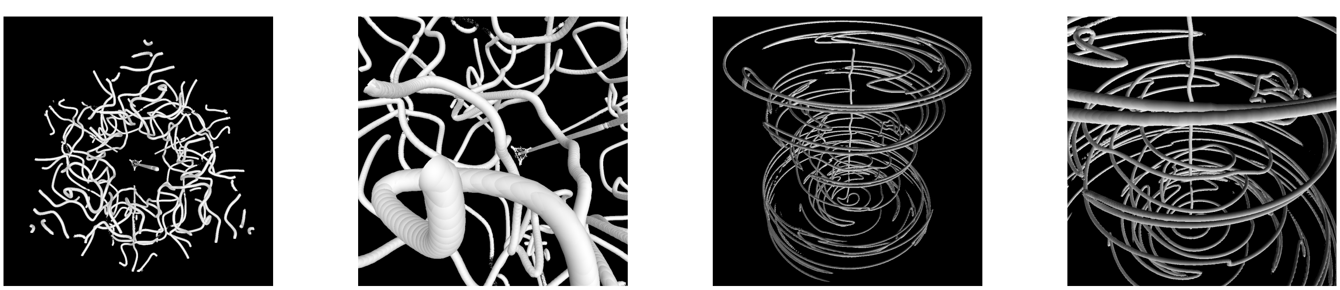


Figure 4: *Performance scaling plot of all datasets using our NVIDIA IndeX implementation (lower is better).*



(a) $3,389$ *vertices, extraction time* $0.061\,sec$ **(b)** $33,619$ *vertices, extraction time* $0.604\,sec$ **(c)** $336,853$ *vertices, extraction time* $6.150\,sec$

Figure 5: *Explicit extraction of point geometry by descending a uniformly distributed set of particles onto the nearest line structure. Here, a CPU implementation of the sectional Newton descent was used.*



(a) *far view* **(b)** *close-up* **(c)** *far view* **(d)** *close-up*

Figure 6: *Here, the camera perspectives are shown that were used for the Nvidia IndeX performance benchmark, showcasing the rendering coverage of the two most challenging data sets under consideration: The* BORROMEAN RINGS *(a), (b) and the* SWIRLING JET *(c), (d). The rendering and volume resolution are the same as given in Table 1 in the main paper.*