Postprint

M. Hlawatsch, F. Sadlo, and D. Weiskopf.
Hierarchical Line Integration.
IEEE Transactions on Visualization and Computer Graphics, 17(8):1148–1163, 2011.
DOI: 10.1109/TVCG.2010.227
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5611509

Hierarchical Line Integration

Marcel Hlawatsch, Filip Sadlo, *Member, IEEE*, and Daniel Weiskopf, *Member, IEEE Computer Society*

***Abstract*— This paper presents an acceleration scheme for the numerical computation of sets of trajectories in vector fields or iterated solutions in maps, possibly with simultaneous evaluation of quantities along the curves such as integrals or extrema. It addresses cases with a dense evaluation on the domain, where straightforward approaches are subject to redundant calculations. These are avoided by first calculating short solutions for the whole domain. From these, longer solutions are then constructed in a hierarchical manner until the designated length is achieved. While the computational complexity of the straightforward approach depends linearly on the length of the solutions, the computational cost with the proposed scheme grows only logarithmically with increasing length. Due to independence of subtasks and memory locality, our algorithm is suitable for parallel execution on many-core architectures like GPUs. The trade-offs of the method—lower accuracy and increased memory consumption—are analyzed, including error order as well as numerical error for discrete computation grids. The usefulness and flexibility of the scheme is demonstrated with two example applications: line integral convolution and the computation of the finite-time Lyapunov exponent. Finally, results and performance measurements of our GPU implementation are presented for both synthetic and simulated vector fields from computational fluid dynamics.**

***Index Terms*— Flow visualization, integral curves, hierarchical computation, FTLE, LIC, GPU.**

I. Introduction

VECTOR fields are involved in a wide range of fields in engineering and science. The spectrum includes mathematics, physics, engineering, and bio-sciences. Visualization is critical for investigating and understanding the intricacy of their spatio-temporal structure, and is subject to extensive, ongoing research.

This paper focuses on techniques that use integral curves, such as streamlines or path lines, to visualize vector fields. Integral curves have the advantage of representing important aspects like flow direction, transport mechanisms, and spatio-temporal relationships. Some of the integral-curve methods such as those related to line integral convolution (LIC) [3] or finite-time Lyapunov exponents (FTLE) [10] require computation of an integral curve for each point inside the domain, covering the domain densely by curves. A trajectory is seeded at each pixel or sampling point and constructed by particle tracing, possibly in both downstream and upstream directions. In addition to particle tracing, further computations along the trajectories may be required. For example, in the case of LIC and its variants, an additional noise texture is filtered by convolution along each trajectory.

The computational complexity of all these methods with dense coverage is $O(N^m K)$ for an m-dimensional spatial domain of resolution N. A factor of N^m is due to the dense sampling of the domain. The factor K represents the number of traditional integration steps along each curve, which is often proportional to the image resolution, in particular for LIC. Especially for (time series of) 3D data and long integration times, common for FTLE computation, this complexity behavior leads to long computation times and sometimes even restricts the user to inappropriately low resolutions.

The motivation for this paper is the observation that, due to coherence, many integral curves partially follow trajectories of different seed points. Our goal is to lower the computational complexity of dense curve integration by reusing parts of the trajectories in the spatio-temporal neighborhood of a solution. To this end, we introduce a new computation scheme for integral curves, or solutions in generic maps, which results in logarithmic effort instead of linear, as for the straightforward approach. In this way, the overall complexity is reduced to $O(N^m \log K)$. Our approach relies on hierarchical construction of the integral curves to allow reuse of already calculated parts; each level of the hierarchy doubles the length of the previously computed curves. We will show that our scheme may be applied to time-independent and time-dependent vector fields alike.

Besides improving the computational complexity, we take into account implementation aspects as well. The algorithm is designed for high intrinsic parallelism performing well on many-core hardware architectures like GPUs. The performance characteristics are documented by scalability experiments with a CUDA implementation. Since discretization on grids is employed for each hierarchy level of the computation, numerical approximation is an issue. The numerical error is analyzed by quantitative error measurements and by deriving the error order with respect to the discretization resolution. We detail how hierarchical computation can be applied to LIC and the FTLE as two typical examples of dense curve integration. For both LIC and the FTLE, extensive performance measurements and several example visualizations are provided. The main benefit of our approach is that dense visualization, e.g., using LIC or the the FTLE, can be generated faster or at higher resolution with the same computation time. Although this comes with increased memory consumption and reduced accuracy, we show that the advantages outweigh the disadvantages in typical applications.

II. Related Work

In general, algorithms in computer science often rely on hierarchical structures or schemes to reduce computational complexity. Typical examples include divide-and-conquer algorithms or dynamic programming. Our approach particularly resembles the acceleration strategy of pyramid algorithms known from digital image processing. Pyramid algorithms were introduced by Burt [2] for image filtering. The idea behind pyramid algorithms is to iteratively construct a pyramid of images with decreasing resolution. For example, Gaussian blur can be efficiently implemented by repeated application of Gaussian filters. Similarly, image operations for computer graphics can be accelerated by

The authors are with the Visualisierungsinstitut der Universität Stuttgart (VISUS), Allmandring 19, 70569 Stuttgart, Germany. E-mail: {hlawatsch,sadlo,weiskopf}@visus.uni-stuttgart.de.

pyramid techniques [23]. The filter principle relies on repeated application of filter operations, which can also be used in the form of general repeated integration according to Heckbert [12]. Due to the direct mapping of regular image grids to textures, pyramid filters lend themselves to efficient GPU implementations; a recent summary of GPU filters is presented by Kraus and Strengert [17]. An application of hierarchical computation in visualization is described by Lum et al. [21], who use a hierarchical method for calculating pre-integration tables for direct volume rendering with $O(N^2)$ complexity, instead of $O(N^3)$ for naive implementations. Although we adopt the acceleration strategy of pyramid and hierarchical methods, our approach exhibits the following structural differences. First, we have to consider filtering along curved 1D trajectories, leading to a more demanding filter process. Second, to increase accuracy, we do not decrease the resolution while progressing to higher levels of the compute hierarchy.

As example applications, we focus on the FTLE and LIC. The FTLE has been known in fluid dynamics literature for a couple of decades [9], [10], [20]. However, most previous work on the FTLE focuses on aspects other than computational complexity. The most prominent examples of FTLE acceleration strategies are due to Garth et al. [7] and Sadlo and Peikert [25], who use adaptive refinement of the computational grids. However, their approach is still subject to linear complexity with respect to integration range, whereas we achieve logarithmic complexity. For the special case of FTLE time series, Brunton and Rowley sketch in a conference abstract [1] that computations from previous time steps can be reused. Our method is more general and allows also the accelerated evaluation of quantities along trajectories, which is demonstrated with LIC in this paper.

LIC is a typical example of texture-based flow visualization. We refer to Laramee et al. [18] for a survey on that topic. Most previous work in that field targets acceleration by mapping to GPUs [14], [16], [33], [36], which does not improve computational complexity. Fast LIC by Stalling and Hege [13], [29] is to our knowledge the only example that reduces the number of LIC filter operations. Similar to our approach, they avoid multiple computation of similar streamlines. However, they scatter streamline information across the grid, whereas we gather that information from lower hierarchy levels. Gathering methods are much more suited for parallel and cache-friendly execution as required by multi-core hardware architectures. Additionally, atomic operations are required for Fast LIC to avoid race conditions between parallel threads. Therefore, Fast LIC is efficient on traditional CPUs, but our hierarchically computed LIC also on GPUs and multi-core CPUs. Furthermore, we support computations beyond LIC, integration of further quantities, time-dependent data, and higher-dimensional domains. Li and Shen [19] use so-called trace slices to accelerate view-dependent texture advection. Similar to our concept of coordinate maps (Section III-A), their trace slices provide a mapping from starting positions to positions on reverse-time path lines. Instead of building a hierarchy with fixed resolution and increasing trajectory length as in our case, Li and Shen create a set with different resolutions similar to mipmapping for multi-resolution texture advection.

III. Hierarchical Scheme

In this paper, we address cases where the computation of densely seeded trajectories, or solutions, is required. The motivation for the proposed acceleration scheme is to exploit the coherence of spatio-temporally proximate portions of these trajectories. In contrast to straightforward approaches, we avoid the computation of similar parts by reusing previously computed partial solutions.

A. Hierarchical Advection

Advection is performed in a hierarchical manner for all points in parallel. Figure 1 (left) illustrates the concept for some seeds contributing to the same solution. In the initial step, a short solution is computed for each seed and the corresponding end point is stored at the seed, possibly together with additional quantities evaluated along the trajectory (Section III-B). This provides a mapping from initial points to their partial solutions, denoted *coordinate map* in this paper. The coordinate map can be seen as a function $\phi_i(x) : D \to D$, with i defining the hierarchy level and typically $D \subseteq \mathbb{R}^n$. Next, a new solution is computed for each point by following this map $s \geq 2$ times, combining the quantities and storing it in the next level of the hierarchy. Subsequent levels are generated by repeating this procedure until the solutions match the prescribed integration range. The construction of the coordinate map for the i^{th} level of hierarchy can therefore be described as $\phi_i(x) = \phi_{i-1}^s(x) = (\phi_{i-1} \circ \phi_{i-1} \ldots \circ \phi_{i-1})(x)$, e.g., $\phi_i(x) = \phi_{i-1}(\phi_{i-1}(x))$ for $s = 2$. From this follows that it is sufficient to store only the current level and to overwrite it with the next level to avoid increasing memory consumption. Of course, the zeroth level, $\phi_0(x)$, has to be derived directly from the underlying problem. In the case of vector fields, $\phi_0(x)$ is computed by integration, i.e., by solving the corresponding initial value problem.

The exponential growth of the integration range with the number of levels in the hierarchy results in logarithmic complexity (see Section III-C), leading to a considerable acceleration of line integration. Additionally, it lends itself to parallel execution on multi-core and many-core architectures like GPUs, due to memory locality and independence of subtasks. In contrast to pyramid methods, which build up hierarchical structures with decreasing resolution from bottom to top, we use hierarchical levels of constant resolution to increase the accuracy of our method and to obtain results of high spatial resolution.

The proposed scheme is applicable to any mapping $D \to D$. In the case of discrete mappings (e.g., arising from graphs), no interpolation is required for following partial solutions; hence the obtained results are exact. If continuous maps, or vector fields, are subject to integration, the coordinate map is discretized by our scheme. In this case, each hierarchical level generally introduces error due to interpolation (Section III-D). Time-dependent

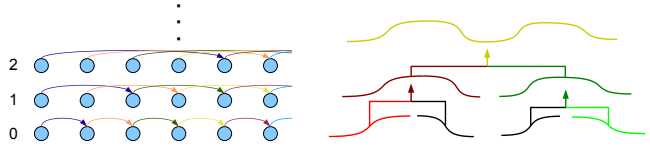


Fig. 1. Left: three levels of the hierarchical computation scheme (with $s = 2$) for points contributing to the same solution. In step 0, short solutions (arrows) are computed from every point. These solutions are stored at their respective starting point. The stored solutions are then used in the next level to construct longer solutions, which are then stored again. This procedure is repeated in the following levels until solutions of designated length are obtained. Right: hierarchical evaluation of quantities—the quantity along a trajectory is computed from the quantities of shorter parts.

mappings, e.g., by path lines in time-dependent vector fields, can easily be handled by our scheme by adding time as a further dimension (Section IV-A).

B. Hierarchical Evaluation of Quantities

This section addresses methods that require not only the computation of end points of trajectories but in addition the evaluation of quantities (performing operations) along them. An example is LIC, where a texture is convolved with a filter whose support spans the trajectory. Our proposed acceleration scheme allows for simultaneous evaluation of such quantities during the hierarchical integration procedure and hence also accelerates their evaluation. Thereby, the quantities are stored with each point in the hierarchical scheme, i.e., in the coordinate map. However, only quantities that can be evaluated in a hierarchical manner lend themselves to acceleration by our scheme. In other words, it must be possible to decompose the operation into subtasks and to hierarchically merge the partial results for the final result (see Figure 1, right). This includes: integration, convolution with specific kernels, computation of extrema (i.e., minimum or maximum values) and average, and distance computation. There, the same operation is applied to the subparts and as merge operation. For the example of maximum values, the maximum of each subsegment of the trajectory is determined first. The merge operation then yields their combined maximum.

Convolution plays a special role. It relies on a signal along the trajectory, which has to be sampled. Our scheme increases the sampling distance and necessarily the size of the convolution kernel with every level of hierarchy. Thus, high spatial frequencies must be suppressed by the kernel to avoid aliasing from undersampling. Direct application of the scheme to perform the convolution would imply a sampling distance in the next level that equals to the size of the filter kernel in the current level. This might result in a too coarse sampling for the convolution. We address this issue by performing the integration of end points and the computation of quantities in a decoupled manner. The quantities are computed by applying the scheme with increased s, i.e., including more samples than for end point computation. This allows for sufficiently large kernel sizes, assuring a proper sampling of the signal (see Section V-B). Another requirement is that the convolution can be decomposed into multiple subsequent convolutions with appropriate filter kernels. These requirements are met by the Gaussian kernel.

Depending on the quantities and the underlying operations, interpolation has to be performed. Accuracy issues resulting from this are addressed in Section III-D.

C. Complexity

The computation of the end point of a trajectory of integration range r and constant integration step size l requires $i_n = r/l$ integration steps in a straightforward approach. In this case, the computational cost depends linearly on r. The number of computation steps in our scheme depends on the number of levels h in the hierarchy and the number of concatenation steps s in each level. Assuming constant s and l leads to

$$r = ls^h \Rightarrow h = \frac{\log \frac{r}{l}}{\log s}.$$

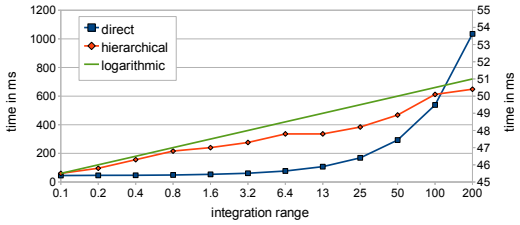


Fig. 2. Execution time in dependency on an exponentially growing integration range with the straightforward (blue, left axis) and the hierarchical (red, right axis) methods (resolution 512^2 and $s = 2$). As a reference, $0.5\log_2$ is plotted (green, right axis). The straightforward method exhibits exponential growth whereas the time for the hierarchical method grows linearly.

Then, the total number of concatenation steps i_h of the hierarchical method is

$$i_h = sh = \frac{s}{\log s} \log \frac{r}{l}.$$

Using the constant $c_h = s/\log s$, we obtain

$$i_h = c_h \log i_n.$$

Therefore, the number of computation steps in the hierarchical scheme grows only logarithmically with increasing integration range. It also follows that s has to be chosen larger than one. Furthermore, i_n and i_h have to be rounded up to the next integer value. Hence, the minimum number of computation steps with the hierarchical scheme is achieved for $s = 2$.

This analysis shows that the hierarchical method already performs only half of the computation steps compared to the straightforward approach when $i_n = 16$. Furthermore, doubling the integration range, i.e., adding another level in the hierarchy, causes only a small constant additional computational cost. Of course in practice, a speed-up of the overall procedure of a factor of 2 is achieved only for considerably larger i_n, due to the setup time required for both methods, e.g., for memory allocation. Figure 2 documents the computation times for a typical example measured with our implementation (see Section VI for implementation details). This performance plot shows that, besides the setup time, the straightforward and hierarchical approaches indeed exhibit the predicted performance characteristics. The plot uses an exponentially scaled x-axis to expose the logarithmic growth of the hierarchical method. Hence, the straightforward method shows exponential and the hierarchical method linear behavior in this plot. Because of a setup time of around 45 milliseconds in this example, a speed-up of two is reached at an integration range around 13, where 128 computation steps are performed with the straightforward and 14 steps with the hierarchical method.

It has to be noted that the theoretical view that both methods perform the same computation steps does not strictly apply (see Figure 3). For the zeroth level, the hierarchical method uses computation steps identical to the straightforward approach, a fourth-order Runge-Kutta integration scheme in this example. All additional levels need only access to single positions for following the coordinate map, necessitating one interpolation operation only. In contrast, fourth-order Runge-Kutta integration, like most other solvers, needs access to several locations within the vector field, involving respective interpolation operations. This means that the acceleration by the hierarchical scheme can benefit not only from a reduced number of computation steps, but also, depending on the hardware and interpolation scheme used, from reduced effort for the computations in higher levels of the hierarchy.

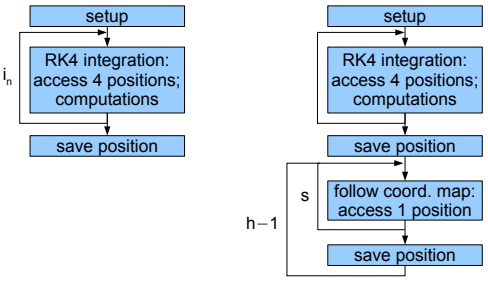


Fig. 3. Flow chart of the straightforward (left) and hierarchical (right) methods with fourth-order Runge-Kutta integration. The straightforward method repeats the same step for the entire computation. Identical steps are executed by the hierarchical method only for the zeroth level of hierarchy. The remaining levels access only single positions in the coordinate map.

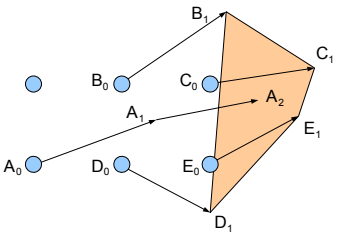


Fig. 4. Linear interpolation inside the coordinate map. The coordinate map is defined on nodes of the sampling grid (blue dots). Position A_0 is mapped to A_1 and likewise $B_0..E_0$ to $B_1..E_1$. Because A_1 does not match a grid point, A_2 is determined by linear interpolation of $B_1..E_1$.

If the integration range does not align with powers of s, the integration range can be matched by adding an additional level of hierarchy together with a reduced step size in the lowest level. The additional level adds only a small overhead in computation time and the smaller step size increases the accuracy of the hierarchical integration.

D. Accuracy

In the general continuous setting, it is unlikely that partial solutions exactly hit nodes of the computational grid. This makes interpolation of the coordinate map necessary (Figure 4). We use tensor-product linear interpolation [6], i.e., bi-linear in 2D and tri-linear in 3D. Unfortunately, interpolation introduces aberration to the solutions computed by the hierarchical method. Therefore, we analyze the introduced error analytically and by measurements for representative datasets.

As derived in the Appendix, the error of our integration scheme is asymptotically bounded by

$$|g_n(x_l)| < \frac{c^2 M}{2L} e^{n2L}, \tag{1}$$

with $g_n(x_l)$ being the global error at node x_l using n levels of hierarchy, L a Lipschitz constant related to the continuity of the flow map, and M the maximum second derivative of the flow maps over all levels of hierarchy. The relevant result of Eq. 1 is that the error is second order in the cell size c.

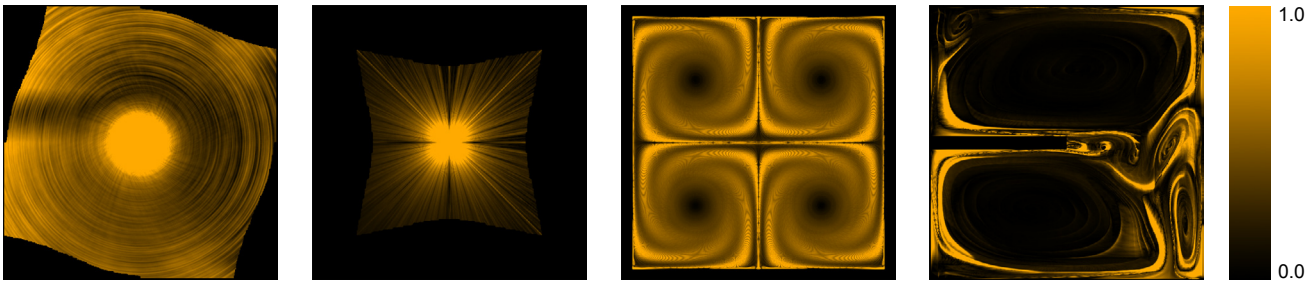
This asymptotic, analytical discussion is backed by a couple of exemplary experimental results. We use 2D vector fields: three synthetic and one dataset from CFD. Motivated by the Helmholtz-Hodge decomposition, two of the synthetic fields are a single whirl and a single source, representing its extreme cases. The third synthetic dataset is the so-called quad-gyre described in [26]. Finally, the buoyancy dataset (Section VI) is a typical CFD example. To restrict the analysis to the error introduced by repeated interpolation of the coordinate maps, the reference end points for the first two synthetic fields are computed analytically as ground truth, and the initial integration in the zeroth level of hierarchy is also performed analytically for these cases.

Figure 5 shows the error distribution for these vector fields. The error is measured as the Euclidean distance between each reference end point and the corresponding hierarchically computed end point. The vector field domain has unit size so that the Euclidean distance can be directly interpreted as distance relative to the extents of the domain. Since the errors of hierarchical integration are small, the error values are scaled in each plot independently for appropriate visualization: the color coding is normalized so that its minimum and maximum colors cover the error values from 0 to the 95$^{\text{th}}$ percentile. The maximum error for the whirl example (Figure 5(a)) is located at the center, which can be explained by the increasing curvature of streamlines toward the center. The error for the source vector field (Figure 5(b)) is much more angle-dependent, but also with highest error at the center. This results from sampling issues with the sampling grid: streamlines are crossing more grid nodes for certain angles, reducing the error introduced by interpolation. In the quad-gyre example (Figure 5(c)), high errors are originating at the borders around the four whirls. The error distribution of the buoyancy dataset (Figure 5(d)) looks similar to the forward FTLE field (Figure 12(d)) of the same dataset, presented in Section VI-B. This can be explained by the fact that the FTLE is a measure for the growth of perturbations (see Section V-A), in our case induced by the hierarchical method through interpolation.

Figure 6 shows a visual comparison of the integration with the straightforward and the hierarchical methods. To avoid visual clutter, only a few results for areas where high deviations occur are visualized. The images show that at most positions with high error, only the last hierarchically computed point deviates substantially from the end point of the reference curve. Still, these end points tend to lie on the reference curves or their continuations. The hierarchical method approximates the direction of the curves accurately in these cases; however, the velocities along the curves tend to be strongly non-linear and are hence approximated with increased error.

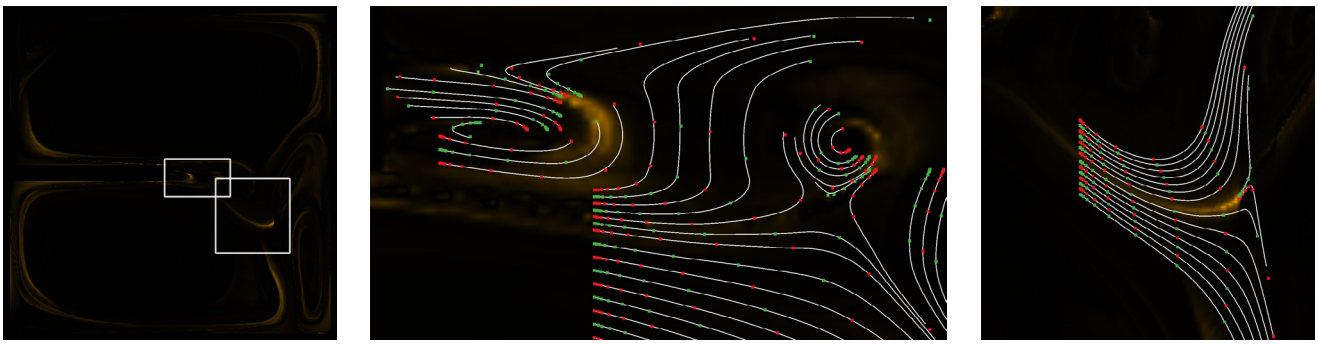
In addition to the error images, we provide a quantitative analysis of the overall error by means of the root mean square error (RMSE) and the 95$^{\text{th}}$ percentile of the Euclidean distance error. Figure 7 shows the error with respect to the resolution of the computational grid. According to this plot, doubling the resolution in each dimension reduces the error approximately by a factor of four. This means that hierarchical advection indeed converges quadratically to the true solution with increasing grid resolution, as predicted by the analytical error bound from Eq. 1. Therefore, numerical accuracy can be balanced with computational and memory costs by adjusting the resolution of the computational grid. Figure 8 additionally shows the error with respect to the integration range. The error depends almost linearly on the integration range.

The computation of quantities along trajectories requires interpolation as well, and thus accuracy issues also occur in this context. An LIC example is shown in Figure 9. Here, the normal LIC implementation produces aliasing artifacts caused by (for this purpose intended) undersampling of the noise texture. These



(a) (b) (c) (d)

Fig. 5. Deviations in hierarchically computed end points (resolution 512^2, 1024 integration steps) for whirl (a), source (b), quad-gyre (c), and buoyancy (d) datasets. The error is normalized with its 95th percentile, which is 0.0023% for the whirl, 0.0012% for the source, 0.0149% for the quad-gyre, and 0.0983% for the buoyancy dataset. A black–yellow color map is applied, with yellow corresponding to the 95th percentile of the error. End points lying outside the dataset domain were rejected from the analysis and colored black. The maximum error is 1.2386% for the whirl, 0.0490% for the source, 0.1594% for the quad-gyre, and 1.2320% for the buoyancy dataset. The average trajectory length (as percentage of the domain size) is 20.48% for the whirl (step size 0.0002), 20.48% for the source (step size 0.0002), 54.18% for the quad-gyre (step size 0.5), and 31.58% for the buoyancy dataset (step size 0.1).



(a) (b) (c)

Fig. 6. Visual comparison of integration in the buoyancy dataset. The zoomed areas ((b) and (c)) are marked in the overview image (a). The white curves are generated with direct integration. The hierarchical method cannot generate a full curve, but only end points; hence, only the end points of every level in the hierarchy are visualized as dots alternately colored red and green for better differentiation of neighboring curves. The lines and dots are seeded in areas with high deviation to demonstrate how the high end point deviation is related to the complete curve geometry. The differential images in the background are normalized with maximum error, which is 1.2320% in this dataset.

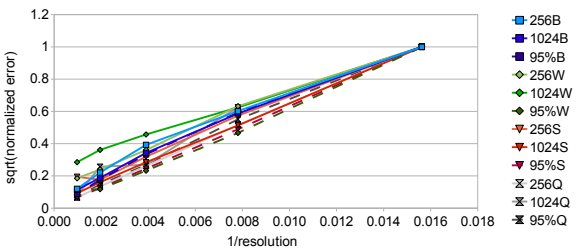


Fig. 7. Square root of the normalized error in dependency on the reciprocal of the resolution in one dimension, for the whirl (W), source (S), quad-gyre (Q), and buoyancy (B) datasets. RMSE for an integration range of 256 and 1024 steps and the 95th percentile of the error for the 1024 case were measured. The error was normalized to the error at reciprocal resolution 1/64.

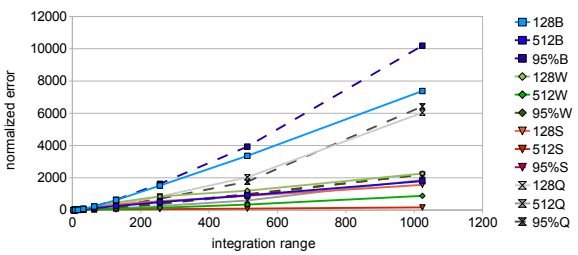


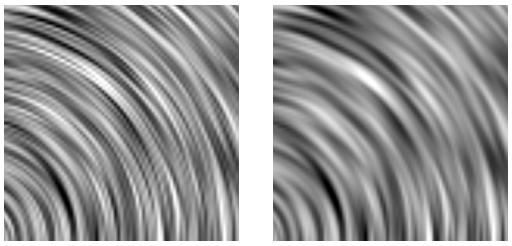
Fig. 8. Normalized error in dependency on integration range. The error was normalized to the error of the shortest range. RMSE at resolutions of 128^2 and 512^2 were measured for the whirl (W), source (S), quad-gyre (Q), and buoyancy (B) datasets. Additionally, the 95th percentile of the error for a resolution of 128^2 was measured.

artifacts are suppressed by the hierarchical LIC method due to multiple resampling of the noise. However, high frequencies are removed not only along the trajectories, but also perpendicular to them, which reduces the detail level of the LIC image. In general, LIC by means of our method is less sensitive to undersampling artifacts of the noise image but at the cost of overall blurring.

As detailed in this section, tensor-product linear interpolation acts as a low-pass filter that removes high frequencies in the quantities and coordinates and thus decreases the accuracy of the results. More accurate reconstruction schemes may be employed to reduce this problem. For example, efficient higher-order B-spline techniques [30], [31] could be used, like the fast third-order interpolation proposed by Sigg and Hadwiger [28], or related pre-filtering methods [4] could be applied.

IV. Algorithmic Details and Implementation

To maximize the speed-up with the hierarchical method, a working set of the involved data must be kept in the fastest available memory, which is often limited in size. For example, implementing the method on GPUs requires the data for computation to be located in graphics memory. Due to space limitations, this can be a particularly important problem for 3D time-dependent data. This memory management issue is discussed in the following together with interpolation accuracy and handling of domain boundaries.



(a) (b)

Fig. 9. Two contrast-enhanced LIC images. Image (a) was computed with the straightforward method. The hierarchical method (b) exhibits less aliasing effects due to blurring caused by repeated resampling via interpolation.

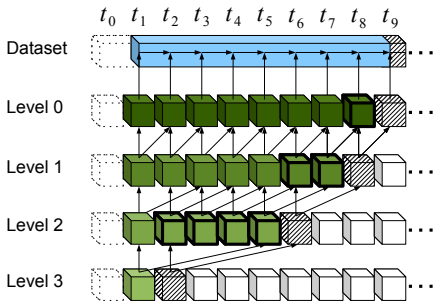


Fig. 10. Hierarchical scheme with time-dependent data: the zeroth level of the hierarchy is derived from the dataset. The green blocks are needed to compute the result (in level 3 in this example) for t_1. The blocks with thick outlines have to be kept in memory to compute the next time blocks (hatched cubes), each overwriting then the block in the respective column. The number of time blocks in memory depends only on the required integration range d.

A. Time-dependent Data

The application of our scheme to time-dependent data can basically be done by treating time as a further dimension of the vector field. Since path lines are equal to streamlines in this space-time field, i.e., they represent solutions to the initial value problem in the respective autonomous system of differential equations, the accuracy of our method is not structurally affected by time-dependency of the data. However, temporal coherence offers the possibility to treat the temporal dimension differently to the spatial dimension regarding memory management, which will be discussed in the following.

We restrict the discussion in this section to the case $s = 2$ (following the coordinate map twice), where the highest speed-up and the lowest memory consumption are achieved (Section III-C). Cases with $s > 2$ can be treated analogously. Further, the integration range is assumed to be an integer multiple of the time span stored in a single time block of the coordinate map in the zeroth level; d denotes the integer multiple. Piecewise linear interpolation in time is additionally assumed.

Spatio-temporal coordinate maps can be mapped to a sequence of spatial coordinate maps. Figure 10 shows how the hierarchy is then built up for time-dependent data (with $d = 8$ in this example). The coordinate map is divided into different time blocks. All solutions saved in one time block belong to trajectories with the same starting and integration time. The number of time blocks determines the temporal resolution of the result computed. For the zeroth level, traditional integration is performed for every time block until the starting time of the following time block is reached. This requirement allows us to follow the coordinate map without temporal interpolation because the time blocks map to temporal positions where another time block is defined. This property is automatically preserved for all levels in the hierarchy. As shown in Figure 10, the computation of a time block for the next level in the hierarchy requires only access to two time blocks of the previous level.

A two-layer streaming approach can be applied to reduce the storage requirements of the hierarchical scheme. We assume two types of memory: faster but smaller M1 memory (e.g., GPU memory) and slower M2 memory (e.g., main memory).

The outer streaming layer handles the streaming of the dataset and the coordinate maps into M2 memory (Figure 10). To compute the first time step of the result, a "triangle" of time blocks (marked green in the figure) has to be calculated. The blocks are computed with increasing starting time (from left to right in the figure). When one time block is computed, the corresponding time span in the dataset can be deleted from memory. When all required blocks of one level inside the "triangle" are completed, blocks of the next level are computed. Every block can replace the block with the same starting time of the previous level. The number of blocks to be computed decreases with every level because of their increasing time span covered. The time block in the final level contains the result and can be streamed out from memory. For the next time step of the result, only one block has to be computed in every level, from the lowest to the highest. With this scheme, only d time blocks (cubes with thick outline in the figure) have to be kept in M2 memory simultaneously and every time step of the dataset has to be streamed to M2 memory only once. This can additionally speed-up computations in large datasets, where only a subset of all time steps fits into memory.

The inner layer of the streaming applies to processing on the faster M1 memory, e.g., computations on the GPU need data in graphics memory. The memory requirements are illustrated in Figure 11. Traditional integration requires to keep two time steps of the dataset with size m_{data} in memory to perform temporal interpolation. The results are stored in memory of size m_{result}. The coordinate map has the same resolution as the result; thus, in the hierarchical scheme, every time block requires m_{result} of memory. Only two of these time blocks are needed for in-core computation and the results can be saved back into the first time block. Hence, the difference in M1 memory consumption can be calculated as $2m_{\text{result}} - (2m_{\text{data}} + m_{\text{result}}) = m_{\text{result}} - 2m_{\text{data}}$. Therefore, the hierarchical method is guaranteed to require less than twice of the M1 memory of the direct method. Even more importantly, the hierarchical computation of time-dependent data needs just the same amount of M1 memory as the hierarchical computation of time-independent data, because, as shown above, time blocks can be overwritten in-place.

Regarding memory I/O operations, the hierarchical method requires more transfer between M1 and M2 memory compared to the straightforward approach. Depending on the hardware and implementation, this might decrease the speed-up gained by the hierarchical method.

The computational complexity for time-dependent data can be derived from Figure 10. For the first time step of the result, the green "triangle" of time blocks is computed. Its size depends on the integer integration range d and it contains $d(\lceil \log_2(d) \rceil + 1) - \sum_{n=0}^{\lceil \log_2(d) \rceil}(2^n - 1) = (d+1)\lceil \log_2(d) \rceil + d - 2(2^{\lceil \log_2(d) \rceil} - 1)$ blocks. Every further time step requires the computation of additional $\lceil \log_2(d) \rceil + 1$ time blocks (hatched cubes in the figure). Thus,

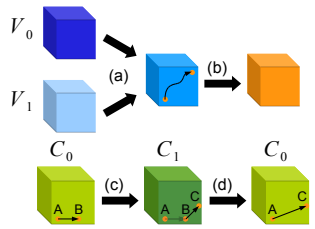


Fig. 11. M1 memory consumption: the direct method must keep two time blocks (V_0 and V_1) of the dataset in memory for temporal interpolation (a). Results are saved in an additional buffer (b). The hierarchical method requires two time blocks (C_0 and C_1) of the coordinate map in memory. Temporal interpolation is not required when following the coordinate map (c). The results are saved back into the first time block C_0 (d).

the computational complexity grows linearly with the number of time steps and approximately logarithmically with the integer integration range d, whereas the direct method grows linearly with both. Because of the overhead for the initial "triangle" in the hierarchical scheme, the method is only faster when several time steps are computed, which is typically the case for time-dependent data. In summary, hierarchical computation is designed for dense computation in space and time.

B. Interpolation

The error introduced by the hierarchical method is caused by the interpolation of the coordinate map; hence the accuracy of the method depends directly on the accuracy of interpolation. This subsection considers the effects of number representation and cancellation [8].

Using relative positions (between end and starting positions) instead of absolute positions in the coordinate map improves the accuracy because then the number representation is not wasted for storing the absolute part of the position, and hence cancellation issues are reduced.

Typical GPUs provide no hardware interpolation with full floating-point precision. In the case of CUDA, hardware interpolation is nowadays only implemented with 9-bit fixed-point precision for the interpolation parameters [22, page 135]. Software interpolation, i.e., using shader or CUDA instructions instead of dedicated hardware interpolation units, therefore additionally improves accuracy.

All results presented in this paper are computed using relative positions and software interpolation. To provide comparisons at full accuracy, the vector field is also software-interpolated during integration with the direct method and in the zeroth level of the hierarchical method. Using software interpolation additionally allows us to transfer the performance gain of the hierarchical method to (parallel) CPU implementations, where interpolation is not supported by dedicated hardware.

C. Domain Boundaries

We distinguish three different spatial domain types when examining trajectory behavior with respect to domain boundaries: periodic, closed, and open domains.

In periodic domains, trajectories outside the domain behave identically to trajectories inside the domain at corresponding locations. This case can be handled by the hierarchical scheme through the usage of relative positions in the coordinate map and their repetition outside the domain. The relative positions include relative periodicity and therefore preserve the correct periodicity of the positions when they are followed. When implementing the method on GPUs, this can be easily handled by using texture wrapping for the coordinate maps. The 3D time-dependent ABC dataset in Section VI is processed with this method.

In closed domains, trajectories always stay inside the domain as in the buoyancy dataset in this paper. This case does not require any dedicated treatment by the hierarchical scheme.

In open domains, trajectories can leave the domain and their behavior outside the domain is not defined by the underlying dataset. This case is difficult to handle because it is not clear how to treat trajectories that leave the domain. For the hierarchical method, these trajectories can influence other trajectories that still are inside the domain because of coordinate map interpolation. A conservative approach is to discard all trajectories leaving the domain as well as the ones influenced by them, similar to the filtering approach in [25]. This can be achieved by adding a border flag in the coordinate map. Trajectories leaving the domain set this border flag; when their positions are used for interpolation, the border flag is propagated. Another approach is to add an additional border with zero vectors around the dataset to approximate stopping trajectories at the domain boundary. The latter approach was used for the overflowed cuboid dataset and both approaches are compared for the breakdown bubble dataset in Section VI.

V. Application

There are several methods requiring dense evaluation of integral curves in vector fields that fulfill the requirements described in Section III-B, and thus can benefit from our acceleration scheme. Two common methods, the computation of the FTLE and LIC, will be discussed in Sections V-A and V-B. Another example is the method by Van Wijk [32], which uses backward tracing of particles to implicitly construct stream surfaces: particles are traced for every point on the domain and hence the method lends itself to acceleration by our scheme. Similar to this, Westermann et al. [37] describe a method for extracting time surfaces in vector fields for visualizing steady flow. Additionally to the backward tracing, their method determines the distances traveled by the particles, which is a quantity suitable for acceleration by our hierarchical scheme. The Mz-criterion [11] for objectively detecting vortices in incompressible flow is another candidate for acceleration.

A. Finite-Time Lyapunov Exponent

Developed for the analysis of the predictability of dynamical systems (i.e., the growth of perturbations), the finite-time Lyapunov exponent [10], [20] is increasingly used as an alternative to the concept of vector field topology [15]. It is defined on a sampling grid that is independent from the underlying vector field and requires the computation of a trajectory for each grid point. For time-dependent vector fields, the trajectories can be streamlines, leading to an instantaneous analysis often similar to that obtained by vector field topology, or path lines, allowing for a time-dependent analysis. The position of the end point of each trajectory is stored inside the coordinate map, also called flow map in this context. The FTLE is then computed from the gradient of the coordinate map.

An important property of the FTLE is that it usually requires sampling grids of high resolution because the topologically relevant features are immanent as ridges, which are often thin and massively folded. Especially in the case of 3D domains, integration of a huge number of trajectories is necessary. Additionally, the trajectories are often required to be comparably long to capture the spatio-temporal structure of the vector field. These requirements make FTLE computation a perfect candidate for acceleration by hierarchical integration.

As the scheme introduces interpolation error (discussed in Section III-D), accuracy can become an issue when the FTLE is used for predictability analysis. However, here we address the usage of the FTLE in the context of a structural analysis of vector fields, where the goal is to extract prominent ridges in the FTLE, namely the identification of coherent regions, and therefore small perturbations or offsets can typically be neglected. We refer the reader to the error analysis in Section III-D and also to the work by Garth et al. [7] for accuracy concerns. In Section VI, our method is applied to the computation of the FTLE and results as well as error analysis are provided.

B. Line Integral Convolution

LIC provides a dense vector field representation by convolving a noise texture along streamlines of a vector field. Usually, box or Gaussian filter kernels (for smoother results) are used. To generate the resulting image, the convolution has to be carried out for every pixel, leading to a dense evaluation of streamlines and therefore qualifying it for acceleration by hierarchical computation. However, as described in Section III-B, the convolution has to be split up into multiple convolutions applied subsequently in our scheme. This is possible if Gaussian filter kernels are used because subsequent convolution with Gaussian filters with widths σ_1 and σ_2 corresponds to a single convolution with a Gaussian filter with width $\sigma = (\sigma_1^2 + \sigma_2^2)^{1/2}$. Hence, application of the hierarchical scheme results in multiple convolution with increasing width of the filter kernel.

In LIC, the convolution of the input noise poses sampling issues. To avoid aliasing artifacts, the noise has to be sampled at an appropriate sampling rate. The sampling rate for the zeroth level of hierarchy depends directly on the input noise. The sampling rate is lowered in subsequent levels by the concatenation during integration with the hierarchical scheme, which requires each convolution to accordingly suppress high frequencies to avoid aliasing. This requirement is met by the Gaussian kernel. With the Fourier transform of the Gaussian filter, $\sigma\sqrt{2\pi}e^{-2\pi^2 f^2 \sigma^2}$, with frequency f, an appropriate sampling can be determined.

Using the Nyquist sampling theorem, a sampling distance of σ leads to a maximum sampled frequency of $f = 1/(2\sigma)$, assuming a band-limited signal. For the non-ideal low-pass characteristics of Gaussian filtering, the sampling distance σ allows us to capture more than 99.99% of the energy in the signal. This is sufficient to avoid sampling artifacts in typical applications. The end points stored in the coordinate map of the previous level of the hierarchy prescribe the minimum sampling distance along a trajectory in the current level. Since we determined σ to be an appropriate sampling distance for convolution, the size of the filter kernel is adapted such that σ of the current level matches this minimum distance from the coordinate map. To get a good approximation of the convolution integral, several samples with this distance along the trajectory are necessary. In our experiments, we use, as noted before, $s = 2$ for the integration of the end points, but for the convolution we obtained good results by integrating until 3σ, i.e., using $s = 6$.

VI. Results

Our implementation uses CUDA 1.0 to perform computations on nVidia GPUs in order to test the suitability of the proposed method for parallel execution. For reference purposes, the implementation of the straightforward approach was also done with CUDA to guarantee a fair comparison. The straightforward implementation and the zeroth level of the hierarchical implementation employ fourth-order Runge-Kutta integration with fixed step size. OpenGL was used for graphical output. All images and measurements were performed with an Intel Core i7 CPU (2.67 GHz), 6 GB RAM, an nVidia GeForce GTX 260 GPU with 896 MB of graphics memory, and Windows 7 Professional.

Our method is evaluated using a time-dependent 2D CFD simulation of buoyant flow, the 3D time-dependent variant of the Arnold-Beltrami-Childress (ABC) analytic vector field from the field of dynamical systems theory, and a time-dependent 3D CFD simulation of a flow over a cuboid. Additionally, a 3D stationary field is included to exemplify the problems with open domains (Section IV-C).

The buoyant flow results from a simulation with a heated boundary at the bottom, a cooled boundary at the top, sidewalls with adiabatic boundary conditions, and gravity acting downward. The container is partitioned by a horizontal divider that forces the buoyant flow to pass on its right side (see Figure 12). The resulting vector field is highly unsteady and exhibits moving vortices and interesting features with respect to Lagrangian coherent structures. The ABC vector field follows the common parametrization of $A = \sqrt{3}$, $B = \sqrt{2}$, and $C = 1$. The second 3D time-dependent example results from a 3D simulation that we call overflowed cuboid. This vector field features a variant of the von Kármán vortex street (Figure 13(e)). At the left end of the volume rendering, there is a cuboid that faces a flow moving from left to right and passing around and over the cuboid. Over time, the vortices of this example tilt from their vertical orientation to horizontal, due to the flow over the upper face of the cuboid. The 3D stationary dataset contains a synthetic model of a vortex breakdown bubble (Figure 14). It represents a perturbed version of Hill's spherical vortex [24].

A. Performance

To analyze the speed-up of real-world applications besides the theoretical estimates from Section III-C, performance measurements of the straightforward and the hierarchical approaches were performed for LIC and FTLE computation. Tables I and II show the corresponding results for the case of 2D time-independent vector fields. The choice of vector field does not affect computation times due to the uniform step sizes. Therefore, the performance numbers are valid for any dataset. We use a grid resolution of 512^2 as a medium-sized example. Computation times are mostly linear in the number of computed texels; thus, timings for other resolutions can be inferred from our measurements. The presented number of steps applies to the straightforward method; the hierarchical method achieves same integration ranges with fewer steps (Section III-C). For both methods, forward and backward advection has to be performed along the streamlines,

TABLE I
COMPUTATION TIMES (IN MILLISECONDS) OF THE STRAIGHTFORWARD (DIRECT) AND HIERARCHICAL METHODS FOR FTLE IN 2D TIME-INDEPENDENT VECTOR FIELDS.

	with setup time			without setup time		
steps	direct	hier.	ratio	direct	hier.	ratio
2×4	46.85	46.36	1.01	0.51	0.42	1.21
2×8	48.94	46.65	1.05	2.41	0.46	5.24
2×16	53.08	47.19	1.12	6.67	0.56	11.91
2×32	60.78	48.11	1.26	13.51	1.11	12.17
2×64	76.27	48.41	1.58	30.08	1.94	15.51
2×128	107.20	48.91	2.19	60.69	2.18	27.84
2×256	168.79	48.99	3.45	121.61	2.75	44.22
2×512	291.89	50.31	5.80	245.05	3.61	67.88
2×1024	538.29	50.10	10.74	491.64	3.99	123.22

doubling the number of computation steps (factor $2\times$ in the column "steps"). Computation times for both the straightforward (direct) and hierarchical methods are given in milliseconds. In addition, the ratio of direct vs. hierarchical methods is included as a measure of speed-up. To compare the computation time alone, values measured without setup time are also presented in the tables. As a consequence, the speed-up by the hierarchical method is much higher in this case.

The times measured for the computation of the FTLE (Table I) can be compared with the measurements in Section III-C. Only a small amount of time is additionally needed to compute the FTLE from the end positions; therefore, FTLE computation and sole construction of the coordinate map consume almost identical computation time. Doubling of the performance is reached at about 100 computation steps—a number of steps at the lower limit for typical computation of FTLE fields. When excluding the setup time, the hierarchical FTLE computation outperforms the traditional computation by more than a factor of two already for only 8 computation steps.

For LIC (Table II), a higher setup time and computation steps with longer execution time can be observed. This is due to additional operations for the convolution, like accessing the noise texture and applying the filter kernel. The performance of LIC is doubled at about 110 computation steps for each direction. When excluding the setup time, hierarchical LIC is already more than twice as fast (compared to traditional LIC) with only 27 computation steps for each direction.

In the case of time-dependent vector fields, the computation of the FTLE is based on path lines. This can also be accelerated with our method by treating time as a further dimension. The memory requirements can be reduced with a streaming approach for the dataset and the coordinate map (Section IV-A). Time measurements (including setup time) for a 2D FTLE field with a spatial resolution of 512^2, 100 time slices, and 512 straightforward integration steps yield: 6.2 seconds for the hierarchical computation compared to 376.5 seconds for the traditional computation. Here, the hierarchical method is almost 61 times faster than the straightforward approach.

Finally, the performance of 3D time-dependent FTLE computation was measured. The FTLE field has a spatial resolution of 128^3, and 64 time steps were computed in all cases. Table III shows the results for increasing integration range. The setup time (around 4 seconds maximum) can be neglected considering the total computation time. As explained in Section IV-A, the hierarchical method has approximately logarithmic complexity with increasing integration range. The direct method theoretically exhibits linear complexity with increasing integration range. However, the results even show a slightly faster growth of the computation time for this method. Our conjectured explanation for this behavior is that the memory locality of nearby starting curves is degrading with longer integration ranges and memory access becomes more expensive due to increased cache misses. The memory consumption measured for the longest integration range corresponding to 64 discrete time steps was around 2500 MB for the hierarchical computation and 440 MB for the direct method, which includes around 400 MB for the dataset. Less than 100 MB of GPU memory is required in both cases.

A comparison with FTLE computation acceleration methods [7], [25] has been left out. In contrast to our method, their performance is data-dependent, making a fair comparison difficult.

Since Image Based Flow Visualization (IBFV) [33] is most popular for fast texture-based vector field visualization, we include a performance comparison between hierarchical LIC and IBFV (see Section VI-B for a comparison of visualization quality). IBFV was implemented in CUDA with per-texel semi-Lagrangian advection in order to obtain performance numbers for the same software and hardware configuration as hierarchical LIC. IBFV implicitly computes an LIC image with an exponential filter kernel [5]. To achieve comparable visual lengths of line structures both in IBFV and LIC, we adjust the exponential filter kernel of IBFV (based on its alpha blending factor) and the width of the Gaussian filter in LIC.

For the example of 2D texture-based visualization of the buoyancy test dataset at an image resolution of 512^2 (Figure 12), we had 6 levels of hierarchical integration equaling 221 straightforward integration steps for LIC, and IBFV with 60 iterations and an alpha blending factor of 0.03. Performance measurements for this example are: 226.76 ms (278.42 ms with setup time) for the

TABLE II
COMPUTATION TIMES (IN MILLISECONDS) OF THE STRAIGHTFORWARD (DIRECT) AND HIERARCHICAL METHODS FOR LIC.

	with setup time			without setup time		
steps	direct	hier.	ratio	direct	hier.	ratio
2×13	63.91	57.08	1.11	12.03	6.26	1.92
2×27	78.13	59.53	1.31	26.01	8.11	3.21
2×55	107.07	61.50	1.74	55.05	8.97	6.14
2×110	163.71	63.04	2.60	112.01	10.69	10.48
2×221	278.42	64.38	4.32	226.76	12.23	18.54
2×443	507.70	66.39	7.65	455.82	14.38	31.70
2×886	966.85	67.77	14.27	914.62	15.83	57.78

TABLE III
COMPUTATION TIMES (IN SECONDS) OF THE STRAIGHTFORWARD (DIRECT) AND HIERARCHICAL METHODS FOR FTLE IN 3D TIME-DEPENDENT VECTOR FIELDS.

integration range	direct	hier.	ratio
2	294	158	1.86
4	587	166	3.54
8	1 198	178	6.73
16	2 494	201	12.41
32	5 407	245	22.07
64	12 337	332	37.16

straightforward LIC method (Figure 12(a)), 12.23 ms (64.38 ms with setup time) for hierarchical LIC (Figure 12(b)), and 40.02 ms (78.87 ms with setup time) for IBFV (Figure 12(c)). The performance measurements show that hierarchical LIC is at least in the same time range as IBFV.

A comparison with Fast LIC [13], [29] is not included because, as stated in Section II, Fast LIC is not suitable for a fast GPU implementation; hence, a fair comparison would not be possible.

B. Visual Results

Figure 12 shows results for the 2D buoyancy dataset. Results for LIC (resolution 512^2, 221 integration steps, step size 0.01), and the computation of the FTLE, with streamlines (resolution 512^2, 1024 integration steps, step size 0.1) and path lines (resolution 512^2, integration range of 10 dataset time steps, step size 0.4), are presented, comparing the reference and hierarchical implementations. The visual quality of IBFV and LIC is also compared. For both cases of the FTLE, two difference images are presented, one comparing directly the FTLE values and one comparing the extracted ridges by their distance. Figures 12(f) and 12(j) show the forward-time FTLE difference, normalized with the 95^{th} percentile of the error (0.47% and 1.16%, respectively). They provide a visualization of the error distribution over the complete dataset. Additionally, a region of interest was selected and the corresponding close-up visualizes the error normalized by the overall maximum (11.66% for the FTLE based on streamlines and 47.33% in the case of path lines).

Compared to the difference of the end points (Section III-D), the FTLE exhibits relatively high differences. This can be explained by the fact that the FTLE is a differential measure (including gradient estimation), and more importantly, it includes a logarithm. Hence, low values are stretched to a larger range, and so is the error. Additionally, the maximum value is decreased and hence normalization accentuates the error. Since the FTLE computed from flow maps (as opposed to evaluation based on renormalization, see [25] for a discussion) is prominently used for the interpretation of the space-time structure of vector fields by means of Lagrangian coherent structures (Section V-A), we use the distance between the resulting ridges as an appropriate error measure, see Figures 12(g) and 12(k). In the first case, the average error (0.014 cells) is far below the cell size, whereas in the second case the maximum error is about half of the cell size. Both close-up regions show the respective region of highest errors where the ridges are consistent. It has to be noted that there are regions of higher error due to inconsistent ridges, meaning that part of the ridge passed the filtering step in one result but not in the other. These cases represent filtering issues, i.e., slight variation of the filtering criterion (here, the minimum required FTLE value) would achieve consistent results locally. All in all, the ridge results are highly consistent regarding the basic difficulty of ridge extraction from noisy and aliased data.

In the hierarchically computed LIC image (Figure 12(b)), the same structure of the vector field can be identified as with the reference image (Figure 12(a)); there are almost no visible differences between the two images. IBFV (Figure 12(c)) generates results of lower visual quality due to inferior low-pass characteristics of the exponential filter kernel (see discussion in [34]) and the repeated resampling (bi-linear interpolation) of the advected image (see discussion in [35]). IBFV requires a number of resampling steps linear with the integration length, whereas hierarchical LIC performs only logarithmic number of resampling steps, leading to much less numerical dissipation. Therefore, hierarchical LIC leads to better visualization quality with at least comparable visualization speed (see Section VI-A).

Furthermore, straightforward and hierarchical FTLE computations are compared for the two examples of 3D time-dependent vector fields (Figure 13). The first row shows the resulting FTLE field for the ABC dataset (straightforward (a), hierarchical (b), resolution 128^3, integration range of 64 dataset time steps, step size 0.05). Spatial periodicity was exploited according to Section IV-C. No differences can be observed by visual inspection and the numerical comparison of the FTLE values reveals that the results are very similar, with a maximum error of 0.86% and a 95^{th} percentile error of 0.18%. Figure 13(c) shows the FTLE difference mapped to the ridge surface of the straightforward result, whereas Figure 13(d) shows the corresponding distance between the ridge surfaces. The aliasing artifacts are due to sharp and high-valued FTLE ridges with respect to the used spatial resolution and appear in both the straightforward and hierarchical computations. Again, the ridges are highly consistent regarding the distance measure.

The results for the overflowed cuboid dataset are presented in the second row. Figure 13(e) shows the straightforward and Figure 13(f) the hierarchically computed FTLE field (resolution 128^3, integration range of 32 dataset time steps, step size 0.05). For this dataset, the maximum difference of the FTLE values is 58.08%, the 95^{th} percentile is 0.79%. Despite the comparably high maximum FTLE differences, the FTLE error is low in the relevant areas—on the ridge surfaces(Figure 13(g)); the spatial deviation of the ridges is negligible in the significant regions (Figure 13(h)).

In Figure 14, problems caused by open domains are shown with the breakdown bubble dataset (resolution 256^3, 1024 integration steps, step size 0.05). As described in Section IV-C, a zero border was added around the dataset. Directly applying the hierarchical method results in a maximum FTLE error of 93.8% and a 95^{th} percentile of 8.1%. The filtering method from Section IV-C reduces the maximum error to 23.3% and the 95^{th} percentile to 0.95%. Although the interior of the bubble exhibits chaotic dynamics, the deviation in terms of the FTLE is negligible there. The apparent error at the axis of the bubble, where folds are generated by thinning and folding, relates to resolution issues due to the finely folded manifold and boundary effects.

VII. Conclusion

We have presented an acceleration scheme for the computation of iterated solutions—or integral curves—for a large field of applications. Additionally, our method can accelerate a certain class of computations of quantities along integral curves. Performance measurements of our implementation confirm the logarithmic computational complexity of the scheme. Since practical implementations are subject to some computational overhead, substantial acceleration can only be expected for medium to long trajectories. In contrast to other acceleration techniques for the computation of integral curves, our method is well suited for modern multi-core or many-core architectures like GPUs. Because every hierarchical computation yields an intermediate result, the user can be provided with a preview at no additional cost. Furthermore, advanced and costly integration methods can be readily integrated in hierarchical computation and affect only the construction of the zeroth level of hierarchy. Therefore, our scheme is particularly efficient for higher-order integration in

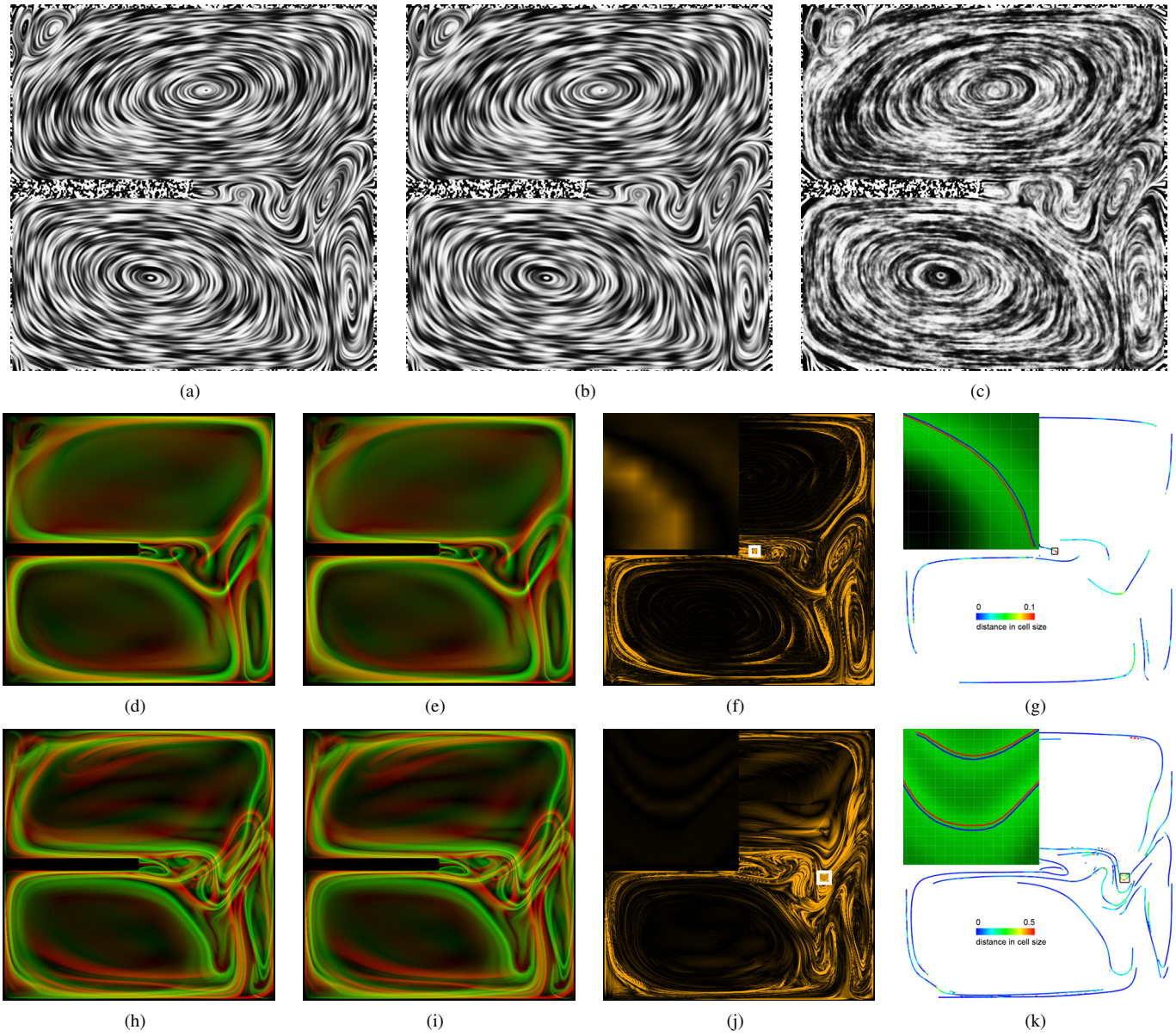
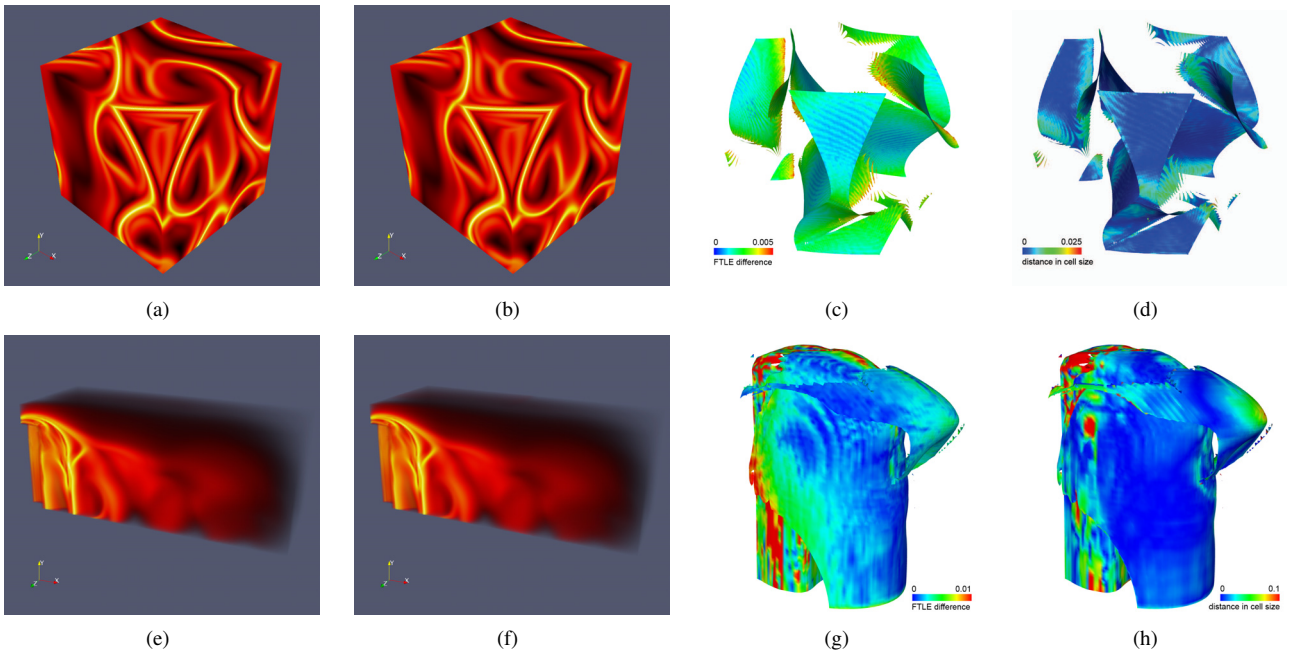


Fig. 12. Results for a single time step of the buoyancy vector field. The first row shows straightforward LIC (a), hierarchical LIC (b), and IBFV (c). Histogram equalization was applied to these images. The remaining images show straightforward ((d) and (h)) and hierarchical ((e) and (i)) FTLE (green forward, red backward). The FTLE in the second row applies to the streamlines in this time step. For the FTLE in the third row, path lines starting at this time step were used. Images (f) and (j) show the forward FTLE error normalized with its 95^{th} percentile. The close-up images of the marked area are normalized with the maximum error. The distance between the ridges from straightforward and hierarchical FTLE computation are presented in (g) and (k) together with a close-up view on the regions of maximum deviation. The red/blue colors in the close-up are used to distinguish FTLE ridges from straightforward and hierarchical computations, respectively; shades of green indicate the FTLE values.

higher-order data. In summary, our method enables the analysis of datasets based on trajectories at a reduced amount of time compared to straightforward methods, especially in the case of high resolutions together with large integration ranges. Considering the gain of computational speed, the drawbacks, loss of accuracy and increased memory consumption, are acceptable for many typical applications.

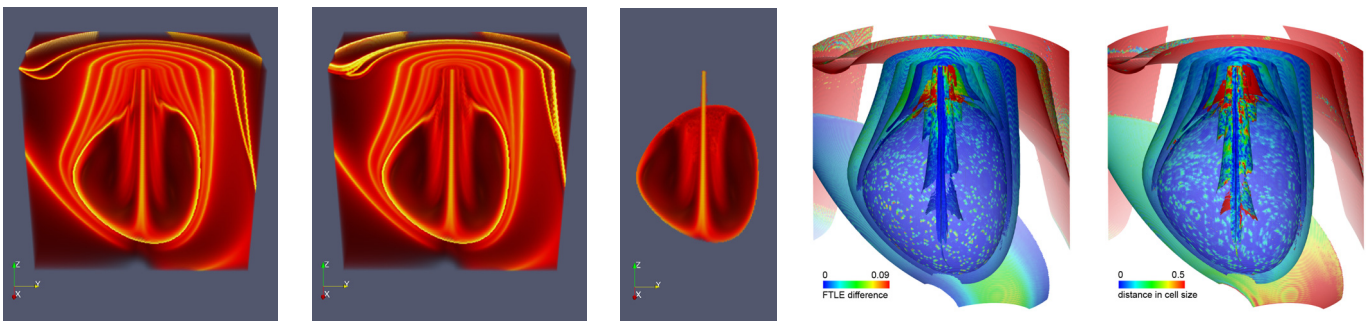
The reason for reduced accuracy is the interpolation of the coordinates when constructing longer trajectory segments from shorter ones. In future work, tensor-product linear interpolation could be replaced by higher-order schemes with better reconstruction quality. We believe that higher-order reconstruction of the coordinate and quantity maps will lead to an integration scheme with better error order. Unfortunately, it is not possible to use adaptive refinement directly with our hierarchical scheme, but a hybrid approach seems promising, where the hierarchical method is used as a first step before adaptive refinement with a direct integration method is applied. The reduced computation time would allow us to compute intermediate results at higher resolutions that then can be adaptively refined. Future research could also investigate applications beyond LIC and the FTLE. For example, stream surfaces, flow level-sets, or the Mz-criterion are expected to benefit from hierarchical computation, and further applications are conceivable. Finally, accelerating the integration in a subarea of the dataset, e.g., for region-of-interest methods or out-of-core computations, is another target of future work.



(a) (b) (c) (d)

(e) (f) (g) (h)

Fig. 13. Results of the 3D time-dependent FTLE computation for two different vector fields. Only the forward FTLE is shown. First row: ABC dataset. The FTLE from straightforward (a) and hierarchical (b) computation is shown using direct volume rendering together with a black-body radiation color map. Image (c) shows the FTLE difference on the ridge surfaces of the straightforward version and image (d) depicts the spatial deviation of the ridge surfaces. Second row: corresponding visualizations of the overflowed cuboid dataset.



(a) (b) (c) (d) (e)

Fig. 14. Results for the breakdown bubble dataset. The left three images show volume renderings of the forward FTLE field obtained with straightforward (a) and hierarchical computation ((b) and (c)). The FTLE field in (c) was filtered so that all FTLE values influenced by the domain boundary are discarded (see Section IV-C). The right two images show the FTLE difference on the ridge surfaces of the straightforward version (d) and the spatial deviation of the ridge surfaces (e).

ACKNOWLEDGMENTS

The authors would like to thank the German Research Foundation (DFG) for financial support of the project within the *Cluster of Excellence in Simulation Technology* (EXC 310/1) at Universität Stuttgart. Many thanks go to Katrin Bidmon for reviewing the appendix.

REFERENCES

[1] S. Brunton and C. Rowley. A method for fast computation of FTLE fields. In *61st Annual Meeting of the APS Division of Fluid Dynamics*, 2008. Electronic reference: `http://meetings.aps.org/link/BAPS.2008.DFD.BR.10`.

[2] P. J. Burt. Fast filter transform for image processing. *Computer Graphics and Image Processing*, 16(1):20 – 51, 1981.

[3] B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In *Proceedings of SIGGRAPH 1993*, pages 263–270, 1993.

[4] B. Csébfalvi. An evaluation of prefiltered reconstruction schemes for volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 14(2):289–301, 2008.

[5] G. Erlebacher, B. Jobard, and D. Weiskopf. Flow textures: High-resolution flow visualization. In C. Hansen and C. Johnson, editors, *The Visualization Handbook*, pages 279–293. Elsevier, Burlington, USA, 2005.

[6] G. Farin. *Curves and Surfaces for CAGD: A Practical Guide*. Morgan Kaufmann Publishers, San Francisco, USA, 2002.

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

[8] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

[9] I. Goldhirsch, P.-L. Sulem, and S. A. Orszag. Stability and Lyapunov stability of dynamical systems: A differential approach and a numerical method. *Physica D*, 27(3):311–337, 1987.

[10] G. Haller. Distinguished material surfaces and coherent structures in three-dimensional fluid flows. *Physica D*, 149(4):248–277, 2001.

[11] G. Haller. An objective definition of a vortex. *Journal of Fluid Mechanics*, 525:1–26, 2005.

[12] P. S. Heckbert. Filtering by repeated integration. *Computer Graphics (Proceedings of SIGGRAPH 1986)*, 20(4):315–321, 1986.

[13] H.-C. Hege and D. Stalling. Fast LIC with piecewise polynomial filter kernels. In H.-C. Hege and K. Polthier, editors, *Mathematical*

Visualization – Algorithms and Applications, pages 295–314. Springer, Heidelberg, Germany, 1998.

[14] W. Heidrich, R. Westermann, H.-P. Seidel, and T. Ertl. Applications of pixel textures in visualization and realistic image synthesis. In *Proceedings of ACM Symposium on Interactive 3D Graphics 1999*, pages 127–134, 1999.

[15] J. Helman and L. Hesselink. Representation and display of vector field topology in fluid flow data sets. *IEEE Computer*, 22(8):27–36, 1989.

[16] B. Jobard, G. Erlebacher, and M. Y. Hussaini. Hardware-accelerated texture advection for unsteady flow visualization. In *Proceedings of IEEE Visualization 2000*, pages 155–162, 2000.

[17] M. Kraus and M. Strengert. Pyramid filters based on bilinear interpolation. In *Proceedings of GRAPP 2007*, pages 21–28, 2007.

[18] R. Laramee, H. Hauser, H. Doleisch, B. Vrolijk, F. Post, and D. Weiskopf. The state of the art in flow visualization: Dense and texture-based techniques. *Computer Graphics Forum*, 23(2):203–221, 2004.

[19] L. Li and H.-W. Shen. View-dependent multi-resolutional flow texture advection. *Proceedings of IS&T/SPIE Visualization and Data Analysis 2004*, 6060:24–34, 2006.

[20] E. N. Lorenz. A study of the predictability of a 28-variable atmospheric model. *Tellus*, 17:321–333, 1965.

[21] E. Lum, B. Wilson, and K.-L. Ma. High-quality lighting and efficient pre-integration for volume rendering. In *Proceedings of Eurographics/IEEE VGTC Symposium on Visualization*, pages 25–34, 2004.

[22] nVidia. CUDA Programming Guide 2.3.1. 2009. Electronic reference: `http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.3.pdf`.

[23] J. Ogden, E. Adelson, J. Bergen, and P. Burt. Pyramid-based computer graphics. *RCA Engineer*, 30(5):4–15, 1985.

[24] R. Peikert and F. Sadlo. Flow topology beyond skeletons: Visualization of features in recirculating flow. In H.-C. Hege, K. Polthier, and G. Scheuermann, editors, *Topology-Based Methods in Visualization II*, pages 145–160. Springer, Heidelberg, Germany, 2009.

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

[26] F. Sadlo and D. Weiskopf. Time-dependent 2-D vector field topology: An approach inspired by Lagrangian coherent structures. *Computer Graphics Forum*, 29(1):88–100, 2010.

[27] H. R. Schwarz. *Numerische Mathematik*. B. G. Teubner, Stuttgart, Germany, 1997.

[28] C. Sigg and M. Hadwiger. Fast third-order texture filtering. In M. Pharr and R. Fernando, editors, *GPU Gems 2*, pages 313–329. Addison-Wesley, Reading, USA, 2005.

[29] D. Stalling and H.-C. Hege. Fast and resolution independent line integral convolution. In *Proceedings of SIGGRAPH 1995*, pages 249–256, 1995.

[30] M. Unser, A. Aldroubi, and M. Eden. B-spline signal processing: Part I–Theory. *IEEE Transactions on Signal Processing*, 41(2):821–833, 1993.

[31] M. Unser, A. Aldroubi, and M. Eden. B-spline signal processing: Part II–Efficient design and applications. *IEEE Transactions on Signal Processing*, 41(2):834–848, 1993.

[32] J. J. van Wijk. Implicit stream surfaces. In *Proceedings of IEEE Visualization 1993*, pages 245–252, 1993.

[33] J. J. van Wijk. Image based flow visualization. *ACM Transactions on Graphics*, 21(3):745–754, 2002.

[34] D. Weiskopf. Iterative twofold line integral convolution for texture-based vector field visualization. In T. Möller, B. Hamann, and R. Russell, editors, *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*, pages 191–211. Springer, Heidelberg, Germany, 2009.

[35] D. Weiskopf, G. Erlebacher, and T. Ertl. A texture-based framework for spacetime-coherent visualization of time-dependent vector fields. In *Proceedings of IEEE Visualization 2003*, pages 107–114, 2003.

[36] D. Weiskopf, M. Hopf, and T. Ertl. Hardware-accelerated visualization of time-varying 2D and 3D vector fields by texture advection via programmable per-pixel operations. In *Proceedings of Vision, Modeling, and Visualization 2001*, pages 439–446, 2001.

[37] R. Westermann, C. Johnson, and T. Ertl. A level-set method for flow visualization. In *Proceedings of IEEE Visualization 2000*, pages 147–154, 2000.

APPENDIX
ERROR ORDER

The error order of our integration scheme is derived according to the typical approach for analyzing explicit single-step integration schemes for ordinary differential equations. We follow the discussion by Schwarz [27]. In our case, iteration over levels of hierarchy is performed in place of integration steps and the error originates from interpolation of the coordinate maps instead of finite integration. We start with the concatenation of the coordinate map $\phi_{i+1}(x) = \phi_i(\phi_i(x))$ (Section III-A). We derive the error order for $s = 2$ (following the map twice) and linear interpolation in 1D space. The extension to n-D space is straightforward by n-D tensor-product linear interpolation. We have chosen linear interpolation because it is a common choice in scientific visualization and presumably results in conservative error bounds as compared to higher order interpolation schemes. We denote error affected coordinate map values by $\tilde{\phi}(\cdot)$ as opposed to exact values $\phi(\cdot)$.

The coordinate map for level $i+1$ at node x_j is obtained by linear interpolation of the coordinate map at level i between the nodes x_l and $x_l + c$ with $l = \lfloor(\tilde{\phi}_i(x_j) - x_0)/c\rfloor$, $x_0 < x_k$ for $(k = 1,\ldots)$, and cell size c:

$$\begin{aligned}\tilde{\phi}_{i+1}(x_j) &= (1-t)\tilde{\phi}_i(x_l) + t\tilde{\phi}_i(x_l + c)\\ &= \tilde{\phi}_i(x_l) + (\tilde{\phi}_i(x_l + c) - \tilde{\phi}_i(x_l))t\end{aligned}$$

with interpolation parameter $t = (\tilde{\phi}_i(x_j) - x_l)/c$.

To simplify the notation, we assume that, for all levels i, $\tilde{\phi}_i(x_j)$ maps to the same cell consisting of its left node x_l and right node $x_l + c$, i.e. $j \equiv l$. This does not affect the analysis of error because the error order is the same for all nodes. This leads to

$$\begin{aligned}\tilde{\phi}_{i+1}(x_l) &= \tilde{\phi}_i(x_l) + (\tilde{\phi}_i(x_l + c) - \tilde{\phi}_i(x_l))\frac{\tilde{\phi}_i(x_l) - x_l}{c}\\ &= \tilde{\phi}_i(x_l) + \Phi(x_l, \tilde{\phi}_i(x_l), \tilde{\phi}_i(x_l + c), c)\end{aligned} \tag{2}$$

with $\Phi(x,y,z,c) = (z-y)\frac{y-x}{c}$. Using Eq. 2, the local discretization error $d_{i+1}(x_l)$ (based on exact values $\phi(\cdot)$) is defined as

$$d_{i+1}(x_l) = \phi_{i+1}(x_l) - \phi_i(x_l) - \Phi(x_l, \phi_i(x_l), \phi_i(x_l + c), c) \tag{3}$$

and the global error is

$$g_i(x_l) = \phi_i(x_l) - \tilde{\phi}_i(x_l). \tag{4}$$

From Eq. 3, the exact coordinate map entry $\phi_{i+1}(x_l)$ can be formulated as

$$\phi_{i+1}(x_l) = \phi_i(x_l) + \Phi(x_l, \phi_i(x_l), \phi_i(x_l + c), c) + d_{i+1}(x_l). \tag{5}$$

Subtracting Eq. 2 from Eq. 5 using Eq. 4 gives

$$\begin{aligned}g_{i+1}(x_l) =& g_i(x_l) + \Phi(x_l, \phi_i(x_l), \phi_i(x_l + c), c)\\ &- \Phi(x_l, \tilde{\phi}_i(x_l), \tilde{\phi}_i(x_l + c), c) + d_{i+1}(x_l)\end{aligned}$$

and extension by $\Phi(x_l, \tilde{\phi}_i(x_l), \phi_i(x_l + c), c)$ leads to

$$\begin{aligned}g_{i+1}(x_l) =& g_i(x_l)\\ &+ \Phi(x_l, \phi_i(x_l), \phi_i(x_l + c), c) - \Phi(x_l, \tilde{\phi}_i(x_l), \phi_i(x_l + c), c)\\ &+ \Phi(x_l, \tilde{\phi}_i(x_l), \phi_i(x_l + c), c) - \Phi(x_l, \tilde{\phi}_i(x_l), \tilde{\phi}_i(x_l + c), c)\\ &+ d_{i+1}(x_l).\end{aligned} \tag{6}$$

To bound the global error $g_i(x_l)$ using the local discretization error $d_i(x_l)$ we have to make sure that $\Phi(x,y,z,c)$ satisfies the

Lipschitz continuity conditions: there is an L such that

$$\left.\begin{aligned}|\Phi(x,y,z,c)-\Phi(x,y^*,z,c)| &\le L|y-y^*|\\ |\Phi(x,y,z,c)-\Phi(x,y,z^*,c)| &\le L|z-z^*|\end{aligned}\right\} x,y,y^*,z,z^*,c\in B \quad (7)$$

inside an appropriate range B for all levels i. We assume the initial coordinate map ϕ_0 to be Lipschitz continuous, a property directly resulting if ϕ_0 is obtained by integration in a linearly interpolated vector field. Then, the first condition in Eq. 7 holds for

$$\Phi(x_l,\phi_i(x_l),\phi_i(x_l+c),c)-\Phi(x_l,\tilde{\phi}_i(x_l),\phi_i(x_l+c),c)$$

because with $y=\phi_i(x_l)$

$$\begin{aligned}\frac{\partial\Phi(x_l,y,\phi_i(x_l+c),c)}{\partial y} &= \frac{\phi_i(x_l+c)-2\phi_i(x_l)+x_l}{c}\\ &= \frac{\phi_i(x_l+c)-\phi_i(x_l)}{c}-\frac{\phi_i(x_l)-x_l}{c}\end{aligned}$$

and $|\phi_i(x_l+c)-\phi_i(x_j)|/|c|\le L'$ due to the continuity assumption on ϕ_0 and because linearly interpolated data (and hence ϕ_i) are Lipschitz continuous, and because $0\le(\phi_i(x_j)-x_l)/c<1$, also for $j\neq l$. The second condition in Eq. 7 holds for

$$\Phi(x_l,\tilde{\phi}_i(x_l),\phi_i(x_l+c),c)-\Phi(x_l,\tilde{\phi}_i(x_l),\tilde{\phi}_i(x_l+c),c)$$

because with $z=\phi_i(x_l+c)$

$$\frac{\partial\Phi(x_l,\tilde{\phi}_i(x_l),z,c)}{\partial z}=\frac{\tilde{\phi}_i(x_l)-x_l}{c}$$

and $0\le(\tilde{\phi}_i(x_j)-x_l)/c<1$ also for $j\neq l$. Using Eq. 7 inside Eq. 6 leads to

$$\begin{aligned}|g_{i+1}(x_l)| \le& |g_i(x_l)|\\ &+L|\phi_i(x_l)-\tilde{\phi}_i(x_l)|\\ &+L|\phi_i(x_l+c)-\tilde{\phi}_i(x_l+c)|\\ &+|d_{i+1}(x_l)|\\ =&|g_i(x_l)|+L|g_i(x_l)|+L|g_i(x_l+c)|+|d_{i+1}(x_l)|.\end{aligned}$$

Using the fact that the error order must be uniform over all nodes (i.e., nodes x_l and x_l+c), we obtain

$$|g_{i+1}(x_l)|\le(1+2L)|g_i(x_l)|+|d_{i+1}(x_l)|.$$

Choosing appropriate constants a and b, the global error satisfies

$$|g_{i+1}(x_l)|\le(1+a)|g_i(x_l)|+b,\qquad(i=0,1,2,\ldots). \quad (8)$$

If $g_i(x_l)$ satisfy Eq. 8, then

$$\begin{aligned}|g_n(x_l)| &\le \frac{(1+a)^n-1}{a}b+(1+a)^n|g_0(x_l)| &(9)\\ &\le \frac{b}{a}(e^{na}-1)+e^{na}|g_0(x_l)|. &(10)\end{aligned}$$

Proof. Here, we repeat the proof by Schwarz [27] for the reader's convenience. Repeated application of Eq. 8 gives the first inequality (Eq. 9)

$$\begin{aligned}|g_n(x_l)| \le&(1+a)|g_{n-1}(x_l)|+b\\ \le&(1+a)^2|g_{n-2}(x_l)|+[(1+a)+1]b\\ &\vdots\\ \le&(1+a)^n|g_0(x_l)|+[(1+a)^{n-1}+\cdots+(1+a)+1]b\\ =&\frac{(1+a)^n-1}{a}b+(1+a)^n|g_0(x_l)|.\end{aligned}$$

The second inequality (Eq. 10) comes from the fact that e^t is convex and therefore for the tangent at $t=0$ the inequality $(1+t)\le e^t$ holds for all t. From this, it follows that $(1+a)^n\le e^{na}$.

□

Using Eq. 10 and assuming[1] $g_0(x_l)=\phi_0(x_l)-\tilde{\phi}_0(x_l)=0$ gives

$$|g_n(x_l)|\le\frac{D(x_l)}{2L}(e^{n2L}-1)\le\frac{D(x_l)}{2L}e^{n2L} \quad (11)$$

with the maximum local error $D(x_l)$ over all levels i.

In order to determine a bound on $D(x_l)$, we will first derive a bound for $d_{i+1}(x_l)$. The idea is to apply the Taylor theorem to ϕ_{i+1} inside Eq. 3 and to determine the remainder. The observation that $(\phi_i(x_l+c)-\phi_i(x_l))/c$ inside $\Phi(x_l,\phi_i(x_l),\phi_i(x_l+c),c)$ represents a discrete derivative with respect to x motivates the reformulation of this term using a true derivative and to bring it into relation with the Taylor expansion.

By the mean value theorem, assuming ϕ_i is differentiable inside $[x_l,x_l+c]$, there is a $\xi\in\,]x_l,x_l+c[$ satisfying

$$\frac{\phi_i(x_l+c)-\phi_i(x_l)}{c}=\frac{\mathrm{d}\phi_i(\xi)}{\mathrm{d}x},$$

resulting in

$$\Phi(x_l,\phi_i(x_l),\phi_i(x_l+c),c)=\frac{\mathrm{d}\phi_i(\xi)}{\mathrm{d}x}(\phi_i(x_l)-x_l). \quad (12)$$

Substituting $\phi_{i+1}(x_l)$ inside Eq. 3 by the Taylor expansion of ϕ_i at ξ (note that ϕ_i is expanded, not ϕ_{i+1}, and the composition $\phi_i(\phi_i(x_l))$ is achieved by evaluating the expansion at $\phi_i(x_l)$) including the remainder term with $\tau=\phi_i(x_l)-\xi$ and $0<\theta<1$ gives

$$\begin{aligned}d_{i+1}(x_l) =&\phi_i(\xi)+\tau\frac{\mathrm{d}\phi_i(\xi)}{\mathrm{d}x}+\frac{\tau^2}{2}\frac{\mathrm{d}^2\phi_i(\xi+\theta\tau)}{\mathrm{d}x^2}\\ &-\phi_i(x_l)-\Phi(x_l,\phi_i(x_l),\phi_i(x_l+c),c).\end{aligned}$$

Using Eq. 12 leads to

$$\begin{aligned}d_{i+1}(x_l) =&\phi_i(\xi)+\tau\frac{\mathrm{d}\phi_i(\xi)}{\mathrm{d}x}+\frac{\tau^2}{2}\frac{\mathrm{d}^2\phi_i(\xi+\theta\tau)}{\mathrm{d}x^2}\\ &-\phi_i(x_l)-(\phi_i(x_l)-x_l)\frac{\mathrm{d}\phi_i(\xi)}{\mathrm{d}x}\end{aligned}$$

and using $\tau=\phi_i(x_l)-\xi$ results in

$$\begin{aligned}d_{i+1}(x_l) =&\phi_i(\xi)+(x_l-\xi)\frac{\mathrm{d}\phi_i(\xi)}{\mathrm{d}x}-\phi_i(x_l)\\ &+\frac{\tau^2}{2}\frac{\mathrm{d}^2\phi_i(\xi+\theta\tau)}{\mathrm{d}x^2}.\end{aligned} \quad (13)$$

Neglecting the last term on the right hand side of Eq. 13, we can identify a first-order Taylor expansion of ϕ_i at ξ with $\phi_i(x_l)$ subtracted. This can be interpreted as the remainder at x_l. The Taylor expansion of ϕ_i at ξ including the remainder term using $\zeta=x_l-\xi$ and $0<\gamma<1$ gives

$$\begin{aligned}\phi_i(x_l)&=\phi_i(\xi)+(x_l-\xi)\frac{\mathrm{d}\phi_i(\xi)}{\mathrm{d}x}+\frac{\zeta^2}{2}\frac{\mathrm{d}^2\phi_i(\xi+\gamma\zeta)}{\mathrm{d}x^2}\\ \phi_i(\xi)+(x_l-\xi)\frac{\mathrm{d}\phi_i(\xi)}{\mathrm{d}x}-\phi_i(x_l)&=-\frac{\zeta^2}{2}\frac{\mathrm{d}^2\phi_i(\xi+\gamma\zeta)}{\mathrm{d}x^2}.\end{aligned} \quad (14)$$

[1] We assume the initial coordinate map to be exact because we aim at analyzing the error introduced by our scheme. However, if $|g_0(x_l)|\neq 0$, the resulting error bound only differs by a constant (Eq. 10).

Substituting Eq. 14 inside Eq. 13 leads to

$$d_{i+1}(x_l) = \frac{\zeta^2}{2}\frac{\mathrm{d}^2\phi_i(\xi+\gamma\zeta)}{\mathrm{d}x^2} + \frac{\tau^2}{2}\frac{\mathrm{d}^2\phi_i(\xi+\theta\tau)}{\mathrm{d}x^2}.$$

From $\xi \in]x_l, x_l+c[$ it follows that $|\zeta| < c$, and additionally from $x_l \leq \phi_i(x_j) < x_l + c$ that $|\tau| < c$, leading to

$$|d_{i+1}(x_l)| < \frac{c^2}{2}\left(\left|\frac{\mathrm{d}^2\phi_i(\xi+\gamma\zeta)}{\mathrm{d}x^2}\right| + \left|\frac{\mathrm{d}^2\phi_i(\xi+\theta\tau)}{\mathrm{d}x^2}\right|\right)$$

$$|d_{i+1}(x_l)| < c^2 \max_{x_l \leq x < x_l+c}\left|\frac{\mathrm{d}^2\phi_i(x)}{\mathrm{d}x^2}\right|.$$

This allows us to bound the maximum local error:

$$D(x_l) := \max_{i=0,\ldots,n\ \ 1}|d_{i+1}(x_l)| < c^2 \max_{\substack{i=0,\ldots,n\ \ 1 \\ x_l \leq x < x_l+c}}\left|\frac{\mathrm{d}^2\phi_i(x)}{\mathrm{d}x^2}\right| =: c^2 M,$$

M being the maximum second derivative inside $[x_l, x_l+c[$ over all levels of hierarchy. From this follows together with Eq. 11 that

$$|g_n(x_l)| < \frac{c^2 M}{2L}e^{n2L}.$$

Hence, the proposed integration scheme is second order in the cell size c.

Marcel Hlawatsch received the Diplom (MSc) degree in computer science from Universität Stuttgart, Germany. Since 2008, he has been a PhD student at the Visualization Research Center, Universität Stuttgart (VISUS). His research interests include scientific visualization, GPU methods, and visualization workflows.

Filip Sadlo received the Diplom (MSc) degree in computer science from ETH Zurich in 2003 where he also did his PhD at the Computer Graphics Laboratory. Since 2008 he has been a research associate at the Visualization Research Center, Universität Stuttgart (VISUS). His research interests include scientific visualization, 3D reconstruction, and imaging.

Daniel Weiskopf received the Diplom (MSc) degree in physics and the PhD degree in physics, both from Eberhard-Karls-Universität Tübingen, Germany, and he received the Habilitation degree in computer science at Universität Stuttgart, Germany. From 2005 to 2007, Dr. Weiskopf was an assistant professor of computing science at Simon Fraser University, Canada. Since 2007, he has been a professor of computer science at the Visualization Research Center, Universität Stuttgart (VISUS) and at the Visualization and Interactive Systems Institute (VIS), Universität Stuttgart. His research interests include scientific visualization, GPU methods, real-time computer graphics, mixed realities, ubiquitous visualization, perception-oriented computer graphics, and special and general relativity. He is member of ACM SIGGRAPH, the Gesellschaft für Informatik, and the IEEE Computer Society.