Output-Sensitive 3D Line Integral Convolution

Martin Falk, *Student Member*, *IEEE*, and Daniel Weiskopf, *Member*, *IEEE Computer Society*

Abstract—We propose a largely output-sensitive visualization method for 3D line integral convolution (LIC) whose rendering speed is mainly independent of the data set size and mostly governed by the complexity of the output on the image plane. Our approach of view-dependent visualization tightly links the LIC generation with the volume rendering of the LIC result in order to avoid the computation of unnecessary LIC points: early-ray termination and empty-space leaping techniques are used to skip the computation of the LIC integral in a lazy-evaluation approach; both ray casting and texture slicing can be used as volume-rendering techniques. The input noise is modeled in object space to allow for temporal coherence under object and camera motion. Different noise models are discussed, covering dense representations based on filtered white noise all the way to sparse representations similar to oriented LIC. Aliasing artifacts are avoided by frequency control over the 3D noise and by employing a 3D variant of MIPmapping. A range of illumination models is applied to the LIC streamlines: different codimension-2 lighting models and a novel gradient-based illumination model that relies on precomputed gradients and does not require any direct calculation of gradients after the LIC integral is evaluated. We discuss the issue of proper sampling of the LIC and volume-rendering integrals by employing a frequency-space analysis of the noise model and the precomputed gradients. Finally, we demonstrate that our visualization approach lends itself to a fast graphics processing unit (GPU) implementation that supports both steady and unsteady flow. Therefore, this 3D LIC method allows users to interactively explore 3D flow by means of high-quality, view-dependent, and adaptive LIC volume visualization. Applications to flow visualization in combination with feature extraction and focus-and-context visualization are described, a comparison to previous methods is provided, and a detailed performance analysis is included.

Index Terms—Flow visualization, visualization techniques and methodologies, three-dimensional graphics and realism, line integral convolution, texture-based vector field visualization, 3D flow, streamline illumination, noise models, GPU programming.

1 Introduction

This paper targets efficient high-quality visualization of 3D vector fields by means of a texture-based representation. The basic strategy of our visualization method relies on line integral convolution (LIC) [6], which serves as a role model for most texture-based vector field visualization techniques. In general, texture-based approaches facilitate a dense vector field representation, reducing the difficulty of finding appropriate seed points for particle tracing. While texture-based methods are well understood for flows on 2D planar domains and on curved surfaces, their extension to 3D domains is challenging. One major issue is efficiency because the computational costs exhibit a naturally cubic increase in complexity with increasing resolution, instead of just a quadratic increase for flows on 2D domains and surfaces. Another fundamental issue of any 3D vector field visualization approach is the perception of the resulting images that might suffer from problems of occlusion, clutter, and inaccurate shape perception.

This paper addresses both the efficiency and presentation issues. Usually, texture-based 3D methods employ a two-step process: first, a volumetric texture representation of the flow is computed; afterwards, this 3D texture is displayed by volume rendering. We propose to tightly link these two steps to avoid cubic complexity. Our starting point is a volume-rendering process that executes an on-the-fly LIC computation only when needed for a respective volume sample point. In fact, most of the volume samples do not contribute to the final image because they are occluded by closer objects, or they are transparent. These sample points are culled by acceleration techniques such as early-ray termination and empty-space skipping, and thus, a costly LIC computation is avoided at those points. Therefore, the rendering costs of our algorithm are largely determined by the complexity of the final image, i.e., the image resolution, the visible depth complexity, and the complexity (length) of LIC streaks. This kind of output sensitivity implies that the rendering speed is mostly independent of the data set size.

The perception issues are addressed by combining a range of known approaches. Occlusion and visual clutter are reduced by feature extraction and focus-and-context visualization that highlight important flow regions and render irrelevant parts less prominent or even completely transparent. In addition, volume clipping allows the user to interactively remove occluding regions. The perception of the shape and orientation of LIC streaks is improved by including illumination. We discuss several illumination models, both for line-based lighting (codimension-2 models) and for gradient-based lighting. In particular, we propose a novel gradient method that achieves high rendering quality without the need for an explicit and time-consuming evaluation of gradients.

Fig. 1 illustrates our 3D LIC method with gradient-based illumination. This data set contains a 3D flow with a vortex structure, where vortices separate from a bottom plate. The focus—the vortex region—is highlighted by green-yellow-red colors, whereas the surrounding context

• *The authors are with Visualization Research Center (VISUS), Universität Stuttgart, Nobelstr. 15, 70569 Stuttgart, Germany. E-mail: {falk, weiskopf}@visus.uni-stuttgart.de.*

Manuscript received 7 Sept. 2007; revised 20 Nov. 2007; accepted 2 Jan. 2008; published online 16 Jan. 2008.
Recommended for acceptance by M. Chen, C. Hansen, and A. Pang.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCGSI-2007-09-0129.
Digital Object Identifier no. 10.1109/TVCG.2008.25.

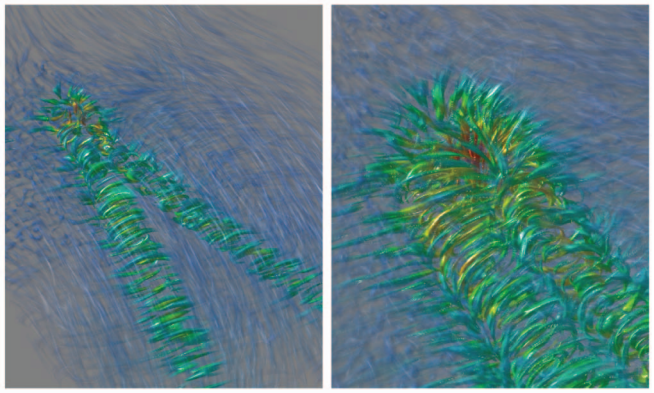

Fig. 1. Example of 3D LIC for a data set with two connected vortex structures, highlighted by green-yellow-red colors: (a) overview image and (b) camera much closer to the connection of the vortex structures (compare with upper left part of (a)).

is shown in almost transparent blue colors. The detailed view with a closed-in camera in Fig. 1b exhibits LIC streaks of much more detail than in the overview in Fig. 1a, demonstrating adaptive rendering.

The main contributions of this paper are

1. a largely output-sensitive algorithm for 3D LIC visualization, including a discussion of principal performance characteristics and acceleration strategies,
2. a prefiltering technique reminiscent of MIPmapping in order to achieve an adaptive, view-dependent, and aliasing-free visualization result,
3. an efficient LIC illumination model that uses precomputed noise gradients to avoid a time-consuming explicit calculation of gradients during runtime, and
4. a fast GPU implementation that supports both 3D texture slicing and single-pass GPU ray casting as volume-rendering techniques.

Our approach is discussed in the context of previous flow visualization approaches, comparing their advantages and disadvantages. In particular, we think that the use of view-dependent computations offers a big advantage compared with previous 3D texture-based methods that rely on precomputation of textures in object space. In addition, our paper emphasizes a proper analysis of our approach: Its computational and memory requirements are discussed in terms of mathematical estimates and actual performance measurements of our GPU implementation; visualization quality is investigated by employing a frequency-space analysis of noise and precomputed gradients.

2 Previous Work

The visualization strategy of this paper adopts and extends 2D LIC [6] in order to show the 3D flow direction by means of a texture-based representation. A large body of previous research on texture-based flow visualization focuses on 2D flow [23]. More recently, texture-based techniques have been extended to the efficient, interactive visualization of vector fields on curved 2D manifolds [22], [24], [25], [26], [27], [41], [47] and on 3D Cartesian domains [39], [48], [49], [50]. All of these recent papers utilize the high processing speed of graphics hardware to accelerate the visualization. Similarly, the implementation of our visualization technique seeks to make efficient use of GPUs. We refer to a book on GPU-based visualization [46] for a comprehensive introduction to GPU methods for texture-based flow visualization.

Another line of research addresses the issues of occlusion, visual clutter, and perception of depth and orientation in the context of flow visualization. For example, interactive clipping of 3D LIC volumes allows users to avoid occlusion problems [32]; selective and perception-oriented rendering can be applied to 3D LIC [16]; limb darkening can be used to improve the perception of LIC lines [13], [14]. All of these implementations of perception-oriented flow visualization are either not interactive or rely on extensive precomputations for particle tracing. The perception of the orientation of thin LIC lines can be facilitated by employing illumination models for objects of arbitrary dimension and codimension [2], in particular for the special case of streamlines [52]. One approach applies streamline illumination to precomputed 3D texture representations of LIC structures or thin threads within the flow [34], [35], [38] or within a tensor field [51]. Another approach applies dynamically computed illumination to the streamlines of finite thickness that result from texture advection [50]. Perceptual experiments show that illuminated streamlines of finite thickness are well suited to promote the perception of spatial orientation [43]. In this paper, we employ the original model for illuminated streamlines [52], as well as an improved lighting model [28] as examples of line-based illumination; in addition, we propose a novel gradient-oriented codimension-1 scheme.

In addition to the LIC component, our work relies heavily on volume rendering to generate final images. We use GPU methods for direct volume visualization: both 3D texture slicing [5] and GPU ray casting [20], [33], [37] are implemented. These basic rendering techniques are enriched by early culling techniques such as early-ray termination and empty-space skipping that mostly follow previous approaches and that are adapted to the requirements of a GPU implementation [8], [18].

3 Output-Sensitive 3D Line Integral Convolution

Three-dimensional flow visualization by LIC consists of two main components: 1) construction of the LIC volume and 2) rendering of the LIC volume. Conceptually, these two components are disjoint and are traditionally implemented as two separate processing steps. We briefly review the basis of 3D LIC and volume rendering before we link them tightly to construct an output-sensitive 3D LIC visualization method.

3.1 Line Integral Convolution

LIC [6], [36] computes

$$\rho(\mathbf{r}) = \int_{s_0-L_s}^{s_0+L_e} k(s-s_0)N(\boldsymbol{\sigma}(s))\,\mathrm{d}s, \qquad (1)$$

evaluating the LIC function value ρ at position $\mathbf{r}$. Vectors and points are denoted by boldface letters and exist in 3D space

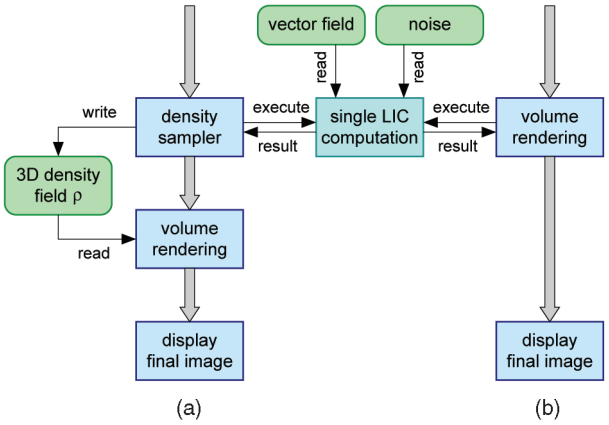

Fig. 2. Comparison of 3D LIC pipelines. (a) The traditional pipeline with an explicit construction of a 3D LIC density field. (b) On-the-fly computation of LIC.

(in case of 3D flow). The function ρ can be regarded as a material density that is subsequently visualized by volume rendering. The input noise is denoted N. The filter kernel $k(s)$ has support $[-L_s, L_e]$; usually, a low-pass filter such as a tent function or a Gaussian function is used. The convolution is performed along the streamline $\boldsymbol{\sigma}(s)$ that is parameterized by arc length s. The streamline is obtained by particle tracing through the vector field $\mathbf{v}$, i.e., by solving the ordinary differential equation:

$$\frac{\mathrm{d}\boldsymbol{\sigma}(\xi)}{\mathrm{d}\xi} = \mathbf{v}(\boldsymbol{\sigma}(\xi)). \tag{2}$$

The initial condition is $\boldsymbol{\sigma}(s_0) = \mathbf{r}$. Curve reparameterization allows us to transform the parameterization according to ξ to an arc-length parameterization according to s. If the vector field is normalized to unit length, $\|\mathbf{v}(\mathbf{p})\| = 1$, then the arc-length parameterization is automatically guaranteed.

In step 2, the volumetric density function $\rho(\mathbf{r})$ needs to be shown on a 2D image plane. The standard approach is to apply volume rendering for this step. We use ray casting or texture slicing as two alternative methods, but virtually, any volume-rendering method could be employed here. Assuming the emission-absorption model [29], volume rendering evaluates the volume-rendering integral:

$$I(t_{\mathrm{end}}) = \int_{t_0}^{t_{\mathrm{end}}} g(\rho(\mathbf{r}(t)))\, e^{-\int_{t}^{t_{\mathrm{end}}} \tau(\rho(\mathbf{r}(\tilde{t})))\,\mathrm{d}\tilde{t}}\,\mathrm{d}t, \tag{3}$$

with the extinction coefficient τ and the emissive source term g, which are determined by the transfer function applied to the LIC density ρ. The variables t and $\tilde{t}$ parameterize a ray through the volume toward the eye, with t_0 and t_{end} corresponding to the volume entry and exit points, respectively. The final radiance that reaches the image plane is denoted $I(t_{\mathrm{end}})$.

As illustrated in Fig. 2a, previous 3D LIC methods compute the 3D volume $\rho(\mathbf{r})$ first and then evaluate the volume-rendering integral in a separate step. Typically, $\rho(\mathbf{r})$ is point-sampled on a uniform grid by calculating (1) at those sample points. During volume rendering, $\rho(\mathbf{r})$ is reconstructed from the discretized, grid-based representation—usually by means of trilinear interpolation. This approach suffers from a number of shortcomings:

1. high memory demands for the uniform grid,
2. additional memory bandwidth requirements for read and write access to the uniform grid,
3. a fixed spatial sampling rate of $\rho(\mathbf{r})$, which does not allow for view-dependent, adaptive rendering (e.g., increasing the spatial resolution in object space for zoomed-in views), and
4. an unnecessary evaluation of the LIC integral for points that are invisible due to occlusion or feature extraction. Usually, features are emphasized by high opacities, whereas uninteresting parts are rendered transparent and could be culled early.

To overcome these issues, we use an on-the-fly computation of the LIC integral (1) during volume rendering, as illustrated in Fig. 2b. This approach has the following advantages:

1. memory for an intermediate grid structure becomes obsolete,
2. the memory bandwidth requirements are dramatically reduced,
3. the sample points can be chosen according to the requirements of volume rendering, allowing for view-dependent and adaptive sampling, and
4. a lazy evaluation of the LIC integral can be used so that the integral will not be computed for invisible sample points.

The disadvantage of the on-the-fly computation is that the LIC volume has to be sampled and evaluated for each frame. The performance gain through lazy evaluation plays a major role in reducing this negative aspect of the on-the-fly computation.

Before we discuss acceleration techniques for volume rendering and the overall rendering speed, we would like to elaborate on what 3D LIC regions will actually contribute to the final image once feature extraction is incorporated. The transfer function g in (3) can be extended so that it depends both on the LIC density $\rho(\mathbf{r})$ and on an additional importance function $\zeta(\mathbf{r})$. We assume that the importance function is defined on the flow domain and that it represents relevant features by high values and unimportant regions by small values. The importance function is application-dependent and could be based on flow characteristics like vortex features, on user-specified features, or on combinations of these. Applications of feature-based 3D flow visualization can be found in [50]. Some examples of importance functions are presented in Section 7.

We recommend that the transfer function g is specified in a way that relevant features are mapped to high opacities, and noninteresting regions are mapped to zero opacity. In this way, only a few, clearly visible depth structures of the flow volume will be rendered to the image plane. Even with perceptually well-designed techniques (see, for example, [15], [16], and the optimizations for two layers [1]), only a few depth layers are easy to perceive. When semitransparent contributions are accumulated over long depth ranges, the final image loses contrast, recognizable depth, and shape information. Here, the volume-rendering integral essentially yields a summation over many independent and

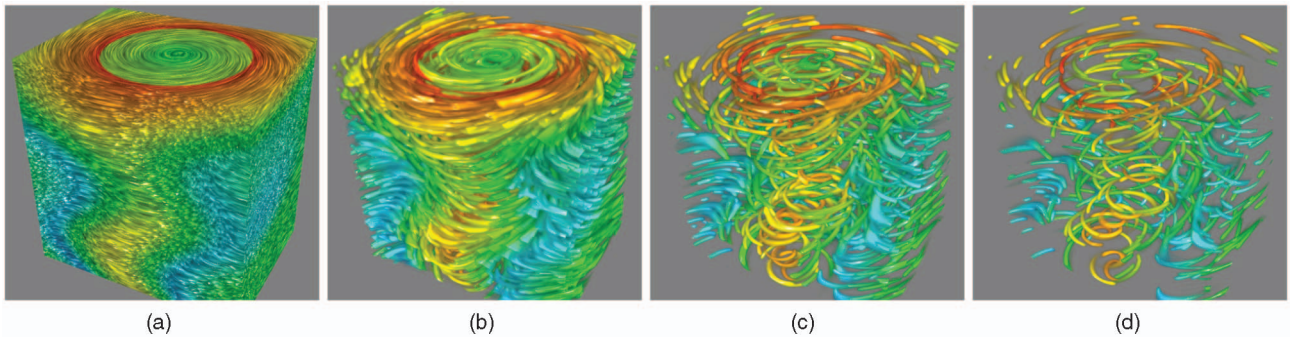

Fig. 3. Different noise models. (a) White noise. (b)-(d) Sparse noise modeled by a Halton sequence with a decreasing number of points.

identically distributed random variables (i.e., the different LIC streaks), which leads to a Gaussian distribution according to the central limit theorem. The width of the Gaussian becomes smaller with increasing effective integration length, resulting in reduced contrast. Therefore, for a perceptually appropriate transfer function, only a few rather opaque volumetric sample positions should contribute to the image.

3.2 Output Sensitivity

The goal of output sensitivity can be formulated as follows: the rendering cost should be proportional to the complexity of the contributions to the information on the image plane and independent of the data set size. The discussion of rendering cost assumes that the LIC and volume-rendering integrals are discretized as Riemann sums based on discrete samples. We propose the following estimate to describe the total rendering cost:

$$C_{\text{total}} = C_{\text{DVR}} M_{\text{DVR}} \cdot C_{\text{LIC}} M_{\text{LIC}} + C_{\text{skip}}. \tag{4}$$

We assume that each sample point along the volume-rendering integral is associated with a cost C_{DVR} and that each LIC integration sample is associated with a cost C_{LIC}. The value M_{DVR} represents the number of relevant volume-rendering samples: visible samples that contribute to the final image. The number of samples along a single LIC integral is denoted M_{LIC}. The term C_{skip} represents the time needed to cull irrelevant volume-rendering sample points via early-ray termination, empty-space skipping, or other related methods.

The cost estimate expects a constant time per volume-rendering and LIC sample. This is a reasonable assumption because each sample is associated with the same number of instructions and memory access operations. In addition, memory access patterns are regular so that cache coherence should be on a uniform level. The cost C_{skip} depends on the algorithms used for culling. Early-ray termination should have constant cost for each volume-rendering sample (independent of data set size) because it only involves one conditional break in the ray-marching loop. The cost of empty-space skipping depends on the data structures representing empty regions. We refer to [18] for a survey of those acceleration techniques. Hierarchical models such as octrees or kd-trees allow for a good scalability with respect to data set size. As explained in Section 6, we use a layered depth structure to determine empty regions in our implementation of GPU ray casting, and the early-z test to skip the LIC computation of empty areas in texture slicing.

In summary, the total rendering cost consists of a part that is of order $\mathcal{O}(M_{\text{DVR}} \cdot M_{\text{LIC}})$ and an offset that accounts for the culling of irrelevant volume-rendering samples. Assuming that the culling cost could be neglected, the render cost of $\mathcal{O}(M_{\text{DVR}} \cdot M_{\text{LIC}})$ corresponds to output sensitivity because M_{DVR} represents the complexity of visible volume points, and the scaling factor M_{LIC} represents the length of LIC lines. Please note that M_{LIC} is influenced indirectly by the image resolution, and M_{DVR} depends on the data set size, which partly break ideal output sensitivity. In practice, the LIC computation is much more time consuming than the volume-rendering part because many LIC integration steps are taken for one volume-rendering step. Therefore, even if the early culling of noncontributing volume samples is suboptimal, the lazy LIC evaluation is extremely beneficial as long as the LIC computation is not executed for noncontributing samples. The detailed performance analysis of Section 7 discusses the actual rendering costs for our GPU implementation.

4 Noise Models and Antialiasing

The noise function $N(\mathbf{r})$ in the LIC integral (1) is instrumental in determining the "look" of the flow visualization and its spatial resolution. The original LIC technique [6] uses white noise to achieve a dense and uniform coverage by LIC streaks. By reducing the density of coverage in the noise function, fewer streaks with a sparser coverage of the flow domain are obtained, as in oriented LIC (OLIC) [44]. Fig. 3 compares white and sparse noise models. Here and in most of the other illustrative vector field visualization examples of this paper, a tornado data set is used.

Both white noise and sparse noise were originally designed for 2D LIC. Our extension of these noise models to 3D targets the following goals: 1) the visualization should be temporally coherent under camera or object motion; 2) the spatial resolution and frequency spectrum of the LIC streaks on the image plane should be invariant under camera or object motion. The first goal targets a flicker-free and coherent exploration of LIC streaks attached to the flow domain. The second goal is connected to adaptive and aliasing-free rendering: a constant spatial resolution on the image plane implies an increasing level of detail of the 3D flow while zooming in; while zooming out, aliasing artifacts are avoided by keeping the spatial frequency spectrum constant on the image plane. We first discuss generic ways of achieving these goals, independent of the density of noise. Later, we describe the differences between sparse and dense noise models.

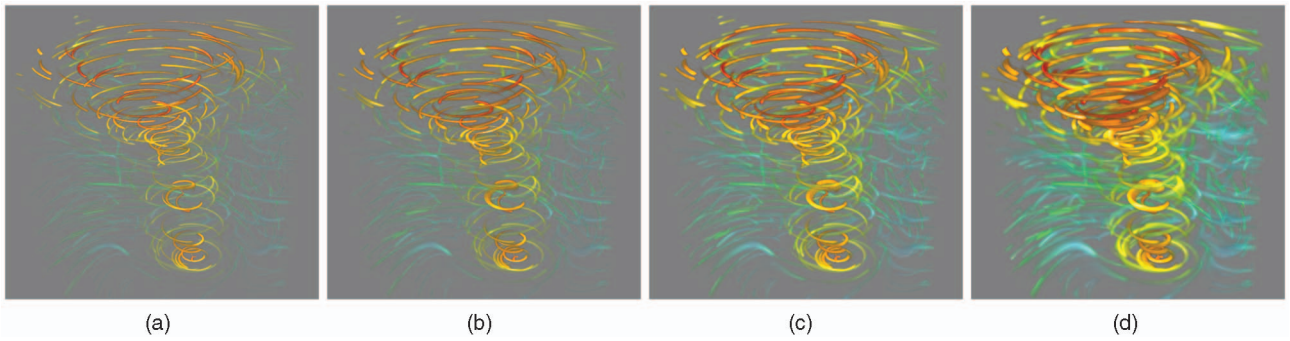

Fig. 4. The thickness of lines is influenced by the width of the filter kernel applied to the 3D noise during noise generation (increasing filter kernel width from left to right).

4.1 Temporal Coherence

The first goal is achieved by using a 3D noise function attached to the 3D flow domain. The noise function is sampled and discretized on a uniform grid, i.e., in the form of a solid noise texture. An on-the-fly construction of the noise function is not fundamentally ruled out, but it is difficult as long as GPUs do not provide procedural random number generators within shaders; in addition, discretized noise textures guarantee temporal coherence because they yield the same value whenever they are sampled at the same spatial position. We use a seamless noise texture with periodic mapping so that a virtually infinite domain is covered by a texture of finite size. Since in our approach, the noise function is at rest with respect to the object space of the flow data set, $\rho(\mathbf{r})$ (see (1)) is at rest with respect to object space because LIC integration and particle tracing are also not altered with respect to object space. As $\rho(\mathbf{r})$ is the basis for rendering, the volume-rendering process generates images of an object that is at rest in object space. Therefore, temporal coherence is guaranteed.

4.2 Invariant Spatial Resolution

The second goal is achieved by a view-dependent filtering of the noise texture. A constant spatial frequency of the noise in object space would lead to a varying spatial frequency on the image plane, after perspective projection. To compensate for the effect of perspective foreshortening, we propose an object-space noise model whose spatial frequency is adapted according to the distance from the camera. Similarly to a prefiltering approach to flow visualization on surfaces [47], we assume that the 3D noise $N'(\mathbf{r})$ can be separated into a sum of octave spectral bands $N_i(\mathbf{r})$:

$$N'(\mathbf{r}) = \sum_{i=-\infty}^{\infty} N_i(\mathbf{r}). \tag{5}$$

According to the Cranley-Patterson rotation [7], different noise functions can be derived from the same template noise by random shifts. The various spectral scales are accommodated by a scaling of the spatial parameters. Then, a spectral band is computed according to

$$N_i(\mathbf{r}) = N_{\text{band}}\Big(2^{(-i)}(\mathbf{r} + \Delta\mathbf{r}_i)\Big), \tag{6}$$

using a single noise template $N_{\text{band}}(\mathbf{r})$ and a random shift $\Delta\mathbf{r}_i$. Furthermore, we assume that the view-dependent noise filtering is narrow-banded and picks out only two neighboring spectral bands (which also makes the energy contribution of the filtered noise finite). Similar to MIPmapping, we model the filtered noise as

$$N(\mathbf{r}) = \sum_{i=i_{\text{depth}}}^{i_{\text{depth}}+1} w_i N_i(\mathbf{r}), \tag{7}$$

where $i_{\text{depth}} = \lfloor \log_2 s_{\text{depth}} \rfloor$. The weights w_i are used to obtain a linear interpolation between the two frequency bands, and s_{depth} describes the depth-dependent scaling according to perspective foreshortening. Fig. 5 illustrates MIPmapping for noise textures applied to a cube.

4.3 Noise Generation

Dense visualization uses a noise template $N_{\text{band}}(\mathbf{r})$ based on white noise. First, white noise is sampled on a uniform grid. The second step applies a band-pass filter. In contrast to convolution in the spatial domain, which is often used for noise construction (e.g., Gaussian [19] or tent functions [40]), we apply an ideal band-pass filter in frequency space: white noise is transformed to frequency space by a fast Fourier transform (FFT), then the filter function is multiplied, and finally, the band-limited noise is transformed back by inverse FFT. In this way, we guarantee accurate control over the noise spectrum.

Sparse noise is modeled according to a sparse distribution of points. The density of the representation can be controlled by adjusting the number of points. A (pseudo) random distribution of those points is not sufficient because it does not guarantee an even coverage by points. A more even distribution of points is achieved by employing low-discrepancy sequences, traditionally used in quasi-Monte Carlo methods [30]. Typical examples include the Halton, Hammersley, van der Corput, Sobol' sequences, or generic (t, m, s)-nets. Please note that for some low-discrepancy sequences care has to be taken so that the Cranley-Patterson rotation does not affect the low-discrepancy property. So far, our implementation uses the Halton sequence, which can be computed fast. Then, the point set is rasterized in a 3D texture. Finally, a filter is applied to smear out the points and reduce the spatial frequency. We use a Gaussian filter because it has the advantage of an exponential fall-off, both in the original space and the frequency space. The filter is applied by multiplication in frequency space. Depending on the filter window of the Gaussian, lines of different thickness can be achieved. Fig. 4 illustrates the effect of four different sized filter kernels applied to the same noise.

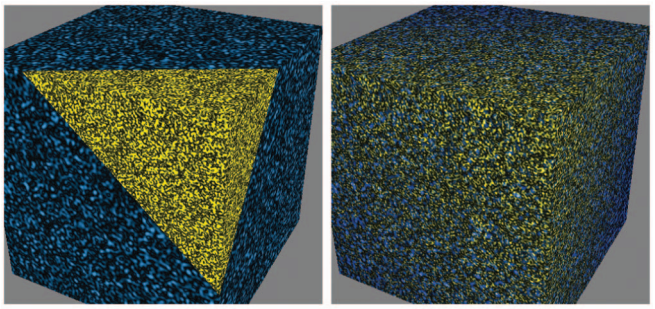

Fig. 5. MIPmapping by combining two neighboring resolution levels that are color-coded as yellow (higher level of detail) and blue (lower level of detail). (a) Without linear interpolation between the levels. (b) With linear interpolation between the levels.

4.4 Frequency Analysis

Independently of the type of noise model, we always generate a precomputed template that is rasterized in the form of a 3D texture. A viewpoint-dependent noise spectrum is obtained through the MIPmapping approach of (7). There are two further sampling components of the overall visualization system that need to be made view-dependent: the sampling of the LIC and volume-rendering integrals. The Nyquist rate for the optimal equidistant sampling of both integrals is directly determined by the noise spectrum. The LIC computation in (1) integrates over the noise function N and, thus, needs to be sampled according to the Nyquist rate of N. The volume-rendering integral works on the LIC result and depends only indirectly on N. LIC reduces the spatial frequency of N only along streamlines, but leaves the frequency perpendicular to streamlines invariant. Therefore, the maximum spatial frequency after LIC is identical to the maximum spatial frequency of the isotropic noise N, and volume rendering should use the same sampling rate. The LIC and volume-rendering sampling rates are modified according to the relative depth-dependent scaling factor s_{depth}. Here, we allow for a continuous, depth-dependent scaling, in contrast to the discrete resolutions in our MIPmapping model. The factor s_{depth} is determined at the respective depth of a volume-rendering sample point. Then, the LIC integration step size is computed as $s_{\mathrm{depth}} s_{\mathrm{LIC}}$, where $1/s_{\mathrm{LIC}}$ is the appropriate sampling rate for the unmodified noise template N_{band}. The multiplication by s_{depth} compensates the scaling of the noise template N_{band} in (6) and (7) so that the noise function N is sampled at the correctly scaled rate.

Appropriate sampling rate for N_{band} typically means the Nyquist rate of an ideally band-limited signal. The template sampling rate might be further modified in order to allow for oversampling to compensate artifacts due to nonideal reconstruction filters. For Gaussian-filtered sparse noise, which is not ideally low-pass filtered, an appropriate sampling rate can be chosen so that most of the energy of the noise is captured by s_{LIC}. Please note that the sampling rate for the LIC integral is not necessarily coupled with the step distance for particle tracing. However, in our current implementation, we have chosen to use the same step size for the LIC integral and the explicit solver for particle tracing so that the explicit solver and LIC integration propagate at the same rate.

Similarly, the volume-rendering integral is sampled with the step size $s_{\mathrm{depth}} s_{\mathrm{ray}}$. Typically, the transfer function for LIC rendering, g, does not contain any high frequencies, and thus, the spectrum of ρ is not modified too much by applying the transfer function (see [3] for a detailed analysis of sampling of the volume-rendering integral). Therefore, the unmodified ray-sampling distance s_{ray} is typically chosen similar to s_{LIC}.

4.5 Adaptive Line Integral Convolution

Usually, the total number of LIC integration steps is a fixed user-specified parameter. In this case, the scheme above leads to LIC streaks of equal length on the image plane, independent of the distance to the camera, because the factor s_{depth} that scales the LIC integration length in object space exactly compensates for the effect of perspective foreshortening. We believe that this behavior of our approach is highly advantageous because it guarantees constant spatial detail and frequency characteristics of the final visualization on the image plane, even if various depth scales of the data set are explored interactively. If required, the LIC integration length could be held constant in object space by adapting the number of integration steps. However, we did not see a useful application for this alternative during our experiments.

The adaptive LIC model allows us to revisit the discussion of output sensitivity from Section 3. According to (4), the main part of the rendering cost is of order $\mathcal{O}(M_{\mathrm{DVR}} \cdot M_{\mathrm{LIC}})$. The adaptive step size for the LIC integral in object space results in projected image-space LIC lines whose lengths are proportional to the number of integration steps M_{LIC}. Identifying the image-space length of streaks with their image complexity, our rendering method is indeed output-sensitive with respect to the LIC line characteristics.

Please note that MIPmapping and the adaptation of LIC integration length affect the density function $\rho(\mathbf{r})$ with respect to object space. When the distance to the camera is changing, the noise changes as well. Similarly, the view-dependent length of LIC streaks in image space implies that there is a different length of integration in object space. In other words, we purposely break the full constancy of $\rho(\mathbf{r})$ in order to achieve antialiasing and constant spatial visualization detail with respect to the image plane. However, the changes of the noise model and LIC integration length are smooth with respect to changing camera distance, and furthermore, they represent just different spatial scale or frequency of the very same underlying model. Therefore, continuous camera or object motion leads to continuous—temporally coherent—changes in the final image.

5 Illumination

So far, we have assumed the emission-absorption model of volume rendering. This section adds volumetric local illumination to improve the shape and orientation perception of LIC streamlines. For example, perceptual experiments demonstrate that illumination applied to streamlines of finite thickness is effective in improving the perception of spatial orientation [43]. Fig. 6 illustrates the effect of lighting: Fig. 6a shows visualization without illumination, whereas Fig. 6b includes lighting. In general, a dense texture-based 3D flow visualization can make use of two classes of local lighting models [50]: codimension-2 illumination and gradient-based illumination.

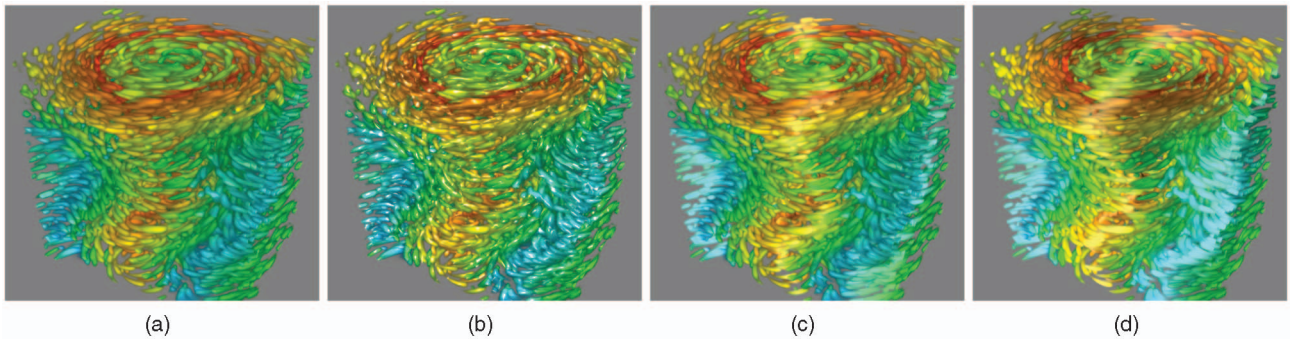

Fig. 6. Comparison of illumination models. (a) Without illumination. (b) Lighting with precomputed noise gradients. (c) Line-based lighting according to illuminated streamlines [2], [52]. (d) Line-based illumination according to Mallo et al. [28].

5.1 Codimension-2 Illumination

The codimension-2 approach treats LIC streaks as infinitesimally thin curves, i.e., as 1D manifolds embedded in 3D space. Objects of a codimension larger than one do not have a unique normal vector, and illumination models for codimension 2 address this issue by computing lighting without the need for an explicitly specified normal vector. In fact, the orientation of a curve is described only through its tangent vector. Since streamlines are by construction tangential to the vector field, the tangent vector of the codimension-2 streaks can be directly accessed from the vector field. Then, curve illumination models such as those by Banks [2], Zöckler et al. [52], or Mallo et al. [28] can be applied. As discussed earlier [50], the main problem of line-oriented volume illumination is that the LIC streaks have finite thickness, and the approximation by infinitely thin curves leads to errors. The visible error can be reduced by applying an illumination transfer function [50], which reduces otherwise artificially large specular highlights in line-oriented lighting. The main advantage of this lighting approach is that it adds only little rendering cost to the emission-absorption model: it just needs access to the vector field at the current volume-rendering sample (the vector field is accessed anyway for the LIC part) and then applies an efficient local illumination computation.

5.2 Gradient-Based Illumination

The alternative approach uses the gradient of the density field $\rho(\mathbf{r})$ as normal vector for local volume illumination. Gradient-based lighting is the standard method for volume rendering in general because it can work with any kind of volumetric object [29]. Therefore, this approach can represent LIC streaks of arbitrary thickness. The main issue, however, is the gradient computation. The texture-advection method in [50] constructs and stores a 3D discretized version of the density field $\rho(\mathbf{r})$ for each frame and then applies central differences to numerically compute the gradient field in a rather time-consuming process. Another related approach precomputes the volumetric density field $\rho(\mathbf{r})$ and the corresponding gradient field and then uses this information for illuminated volume rendering during runtime [38]. All methods that compute and store gradients on a 3D grid suffer from an increased memory footprint for the gradients. In addition, the quality of gradient reconstruction tends to be suboptimal because of the typically low spatial resolution of the grids and because of the approximation by central differences.

Since our 3D LIC strategy builds upon the idea of lazy evaluation, a brute-force computation of gradients on a full grid is not appropriate. One alternative is to evaluate central differences on the fly when they are needed for a visible volume sample: the LIC integral (1) is computed for six neighboring positions to determine the gradient, i.e., the number of LIC computations is increased by 600 percent. Since the LIC part consumes a large part of the overall rendering costs, the visualization speed is significantly reduced.

Therefore, we propose a novel gradient scheme that makes use of partly precomputed gradients to avoid the evaluation of neighboring LIC positions. Assuming straight streamlines $\boldsymbol{\sigma}(s)$ in (1), the gradient can be determined by

$$\begin{aligned}\nabla_{\mathbf{r}}\rho(\mathbf{r}) &= \nabla_{\mathbf{r}} \int_{L_s}^{L_e} k(s) N(\boldsymbol{\sigma}(s+s_0))\,\mathrm{d}s \\ &= \int_{L_s}^{L_e} k(s)\,\nabla_{\mathbf{r}} N(\boldsymbol{\sigma}(s+s_0))\,\mathrm{d}s, \end{aligned} \tag{8}$$

because the gradient operator $\nabla_{\mathbf{r}}$, being a derivative operator, commutes with integration, and the filter kernel $k(s)$ is independent of position $\mathbf{r}$. Equation (8) states that the gradient for volume illumination can be computed by evaluating the LIC integral over the gradients of the noise N. The noise function can be precomputed and so can the noise gradient. The tuple $(N, \nabla_{\mathbf{r}} N)$ can be stored in a joint 3D texture with four data components: one scalar noise component and three gradient components. Equations (1) and (8) can be integrated simultaneously over the combined $(N, \nabla_{\mathbf{r}} N)$ field because both integrals are defined over the same streamline domain.

We have assumed straight streamlines to rule out that varying streamline positions $\boldsymbol{\sigma}(s)$ would indirectly affect the gradient operator. For curved streamlines, the gradient operator's direction within the LIC integral follows a co-moving Frenet frame [11]. This can be accommodated by rotating the accessed gradient $\nabla_{\mathbf{r}} N$ while transporting it along the streamline. However, the gradient of short and medium-length LIC streaks can be well approximated by applying (8) without considering additional rotations, because of the following reasons. First, the shorter a LIC streamline, the less deviation in the orientation of the Frenet frame is to be expected. In addition, the filter kernel usually weighs more distant sample positions less pronounced. Furthermore, for a streamline with constant curvature and torsion, the less accurately approximated gradient contributions are essentially canceled for a symmetric filter because the noise gradients are rotated in

opposite directions for the positive and negative LIC integration directions.

Another consideration is the appropriate sampling rate for integrating over $\nabla_{\mathrm{r}} N$ because the gradient operator changes the spectrum of N. In general, the Fourier transform $\mathcal{F}$ of the partial derivative of a function $f(x,\ldots)$ is $\mathcal{F}[\partial f(x,\ldots)/\partial x](\omega,\ldots) = \mathrm{i}\omega\mathcal{F}[f(x,\ldots)](\omega,\ldots)$, where x and ω are coordinates in the original and the Fourier spaces, respectively. Accordingly, the magnitude is given by $\|\mathcal{F}[\partial f(x,\ldots)/\partial x](\omega,\ldots)\| = |\omega| \cdot \|\mathcal{F}[f(x,\ldots)](\omega,\ldots)\|$. Therefore, the Nyquist rate for ideally band-limited noise, such as our dense noise model in Section 4, is not changed by the gradient operator because the multiplication by $|\omega|$ does not affect the support of a function (except for the DC component at $\omega = 0$). Gaussian-filtered noise, such as the sparse noise model in Section 4, does not have a compact support in its spectrum. Therefore, the multiplication by $|\omega|$ modifies the full range of frequencies. However, the Gaussian falls off exponentially in frequency space, mostly overruling a multiplication with a linear function. Thus, the cutoff frequency that contains most of the spectral energy is similar for N and $\nabla_{\mathrm{r}} N$.

5.3 Summary and Comparison

In summary, our method of precomputed noise gradients allows us to determine a gradient-based on-the-fly illumination with little additional cost: the same LIC integration with the same sampling rate is applied to both N and $\nabla_{\mathrm{r}} N$ to simultaneously compute high-quality gradients that are not affected by reconstruction problems which might occur for central-differences approaches. The additional memory required for the precomputed $\nabla_{\mathrm{r}} N$ can be kept small because periodicity is exploited for both N and $\nabla_{\mathrm{r}} N$. Therefore, gradient-based illumination comes at no extra cost, except for slightly higher memory bandwidth requirements in order to access $\nabla_{\mathrm{r}} N$. This additional cost is small because GPUs provide efficient texture read for such a cache-coherent access scheme.

We recommend using either line-oriented illumination methods (for very thin, possibly long LIC lines) or the method of precomputed gradients (for moderately thin and thick lines of shorter length). Example images of different lighting approaches are compared in Fig. 6.

As discussed in [50], in the context of 3D texture advection, an illumination transfer function can also be applied to 3D LIC in order to improve the appearance of line-oriented illumination by utilizing specular edge darkening. Similarly, any of our 3D LIC rendering methods can imitate a halo effect around lines based on limb darkening [13], regardless of whether they employ any illumination model or not. Halos together with properly chosen line colors (via transfer function) and line lengths lead to an improved visual perception of line continuity, which is an important element in understanding the spatial structure of those lines [16]. Fig. 7 illustrates the effect of those techniques.

6 Implementation

Our model of 3D LIC leads to completely independent computations of the LIC integral for each volume sample point. Therefore, this approach is perfectly suited for the highly parallelized hardware architecture of today's GPUs. This section focuses on our GPU implementation of 3D LIC; a corresponding CPU implementation should be straightforward. We use C++ as programming language and OpenGL in combination with ARB vertex and fragment programs for graphics programming. Support for Shader Model 3.0 is expected for functionality such as dynamic flow control for loops and branching. The CPU-based Fourier transformations are provided by the FFTW library.[1] The code was tested on PCs with Linux and Windows XP operating systems and with NVIDIA GeForce 6xxx, 7xxx, and 8xxx GPUs.

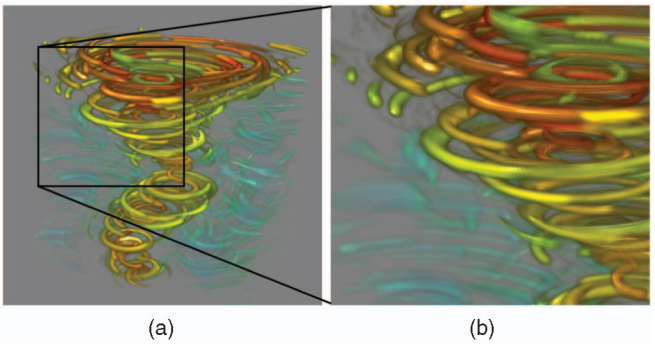

Fig. 7. Visual perception of line continuity is improved by halos and appropriate line colors and line lengths. (a) Overview image. (b) Enlarged details.

The core rendering part of our system can be separated into two connected components: volume rendering and 3D LIC computation. For volume rendering, we have implemented both 3D texture slicing and GPU ray casting [8]. Front-to-back volume traversal is applied in both cases. The fragment programs for volume rendering are slightly modified: instead of accessing a scalar-field texture, the LIC integral (1) is evaluated to compute the density ρ. We discuss the details of the different visualization components in the following.

6.1 Line Integral Convolution Computation

The LIC computation requires access to the vector field and the noise template. Both fields are held in 3D textures. The vectors are decomposed into a normalized direction (three Cartesian components) and their magnitudes (one component), and they are stored together either in RGBA 8-bit, RGBA 16-bit floating-point, or RGBA 32-bit floating-point format. For unsteady flow, the vector field texture is updated for each frame, which eventually leads to an animation with varying streamlines.

The noise is stored in a single-component 8-bit texture. If the method of precomputed gradients is used, this texture is extended to RGBA 8-bit format to also hold the precalculated gradients of the noise. The discretized LIC integral is evaluated within two fragment shader loops: one loop for the positive direction of the streamline, the other for the negative direction. During each iteration, the vector field and noise textures are accessed at the current particle location in object space. The noise value is weighted by the filter kernel and accumulated in the density ρ. The filter function is implemented as 1D texture to allow for arbitrary tabulated functions. We recommend

1. http://www.fftw.org.

using a Gaussian filter for optimal filtering quality if a symmetric filter kernel is wanted. For asymmetric filtering in the sense of OLIC, we recommend a piecewise Gaussian filter: one half with short fall-off distance and the other half with long fall-off distance. The vector information is used for particle tracing according to an explicit integration scheme for Lagrangian particle tracing. We apply a first-order Euler integration but that could be replaced by a higher order Runge-Kutta method if needed. In our current implementation, we have chosen to use the same sampling rate for the LIC integral and particle tracing in order to concurrently sample the particle path. However, the step size for particle tracing and LIC integration could be decoupled as long as the particle tracer provides a means of accessing equidistant positions along the trace as basis for LIC integration. In our implementation, the step size is computed as $s_{\text{depth}} s_{\text{LIC}}$, where s_{depth} is determined at the respective depth of a volume-rendering sample point (see Section 4).

After the shader loops, the density ρ is mapped via a transfer function to RGBA values for volume rendering. We employ an extended transfer function to additionally visualize flow features and other attributes by color coding. Conceptually, our transfer function depends on three parameters: ρ, one vector field attribute α (i.e., the magnitude or one of the vector components), and the value of the importance function ζ. A separable function model allows us to split this 3D transfer function into several 1D transfer functions. First, α is mapped to RGB colors. Then, the density ρ is mapped to opacity and a relative luminance that is multiplied with the RGB color from the first step. This mapping is used to extract clearly visible streamlines. Here, an appropriate choice of parameters facilitates limb darkening [13] and an optional illumination transfer function [50]. Finally, the importance value ζ is mapped to a weighing factor that is multiplied with the RGBA value from step two. All three 1D transfer functions are implemented as 1D dependent textures. The first and third steps are optional. The importance function ζ can be represented by an additional precomputed 3D texture. A typical example of a flow feature is the λ_2 value [17]. Often, vector magnitude is a sufficient importance measure; then, no additional importance texture is needed because the vector magnitude is attached to the vector field texture already. For an overview of flow features, we refer to the survey article [31]. Additionally, we support clipping planes to allow the user to explore the interior of a dense LIC volume by cutting away otherwise occluding parts [32].

The RGBA value from the 3D transfer function may be directly used for compositing during volume rendering. Optionally, local illumination is applied within the fragment program to modify the RGB value. Our implementation supports line-based illumination according to Banks [2], Zöckler et al. [52], and Mallo et al. [28], as well as gradient-based illumination with precomputed noise gradients.

6.2 Volume Rendering

One of our volume-rendering approaches is 3D texture slicing. Standard texture slicing is modified in the following aspects: 1) the volume-rendering fragment shader executes the aforementioned LIC computations, transfer function lookups, and illumination computations; 2) early-ray termination is included; 3) transparent fragments are culled early to avoid the LIC computation in empty space. Aspects 2 and 3 utilize the early-z test to efficiently cull unnecessary fragments, skipping the execution of their shader programs. The depth buffer is used as a mask for culled fragments. This depth mask is set by an additional render pass that is executed once per slice and whose fragment program checks whether the sample point is transparent (according to the importance volume and its transfer function) or whether the opacity threshold has been exceeded. Accordingly, the depth value is set to zero when the fragment can be culled, otherwise to one. Front-to-back compositing of subsequent slices is performed by blending within a framebuffer. Our implementation supports buffers with both 8-bit fixed-point format (i.e., the standard framebuffer) and 16-bit floating-point format (with framebuffer objects (FBOs)) for higher accuracy. Slicing artifacts are reduced by employing Monte Carlo methods. The sampling positions are shifted along each viewing ray by an offset. This offset is constant per ray and stored in a texture covering the image plane. The offset texture is filled with white noise and scaled so that the offset is at most as large as the sampling distance in the volume.

As an alternative volume-rendering technique, GPU-based single-pass ray casting is employed using the ray-casting framework by Stegmaier et al. [37]. Monte Carlo methods are again used to reduce sampling artifacts as mentioned above. Starting at the front-faces of the volume's bounding box, the ray-casting algorithm traverses the volume within a fragment-shader loop. Early-ray termination is supported in the form of a conditional break in the shader loop. For empty-space skipping, we propose an approach that we call layered empty-space skipping, which is a reminiscent of depth peeling [9]. The idea is to build a layered depth structure that partitions the space into equidistant slabs parallel to the image plane. Each layer contains the number of steps through empty space from the beginning of the layer, indirectly representing this empty space. We use a separate ray-casting rendering pass to determine this number of steps. This rendering pass traverses the volume to check for visibility according to the importance function. As soon as the shader hits a nontransparent sample, it is aborted, and the number of steps traversed so far is written into the layer texture. In the actual LIC ray-casting pass, these layer textures are used to offset the starting positions of each ray in order to skip the empty space. In addition, the LIC ray-casting shader checks for zero opacity of the importance function before it initiates a LIC integration. In this way, the LIC computation is skipped for all empty regions even if they are not represented in the layered depth structure (which can happen if there is a nontransparent area between the beginning of the depth layer and the current, transparent sample point). Volume clipping requires special considerations for GPU ray casting. Since we only render the bounding box of the volume to initiate ray casting, we can use the hardware clip planes to cut this bounding box. The resulting hole in the surface of the bounding box is then covered by an additional polygon (i.e., the intersection between the clipping plane and the bounding box) to ensure that rays are generated in cut-away parts of the volume. The source code of a simplified version can be found on the project Web page.[2]

2. http://www.vis.uni-stuttgart.de/texflowvis.

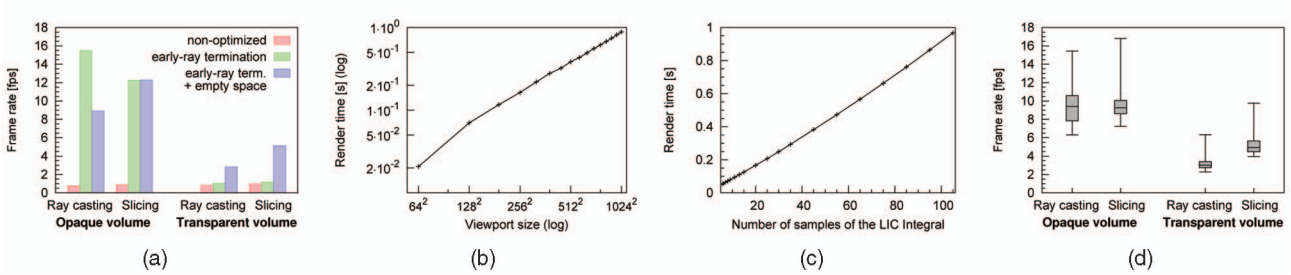

Fig. 8. Rendering performance and output sensitivity. (a) Comparison for different rendering options. (b) Dependency on the viewport size. (c) Increasing number of samples for the LIC integral. (d) Box plots of the performance statistics for varying camera positions.

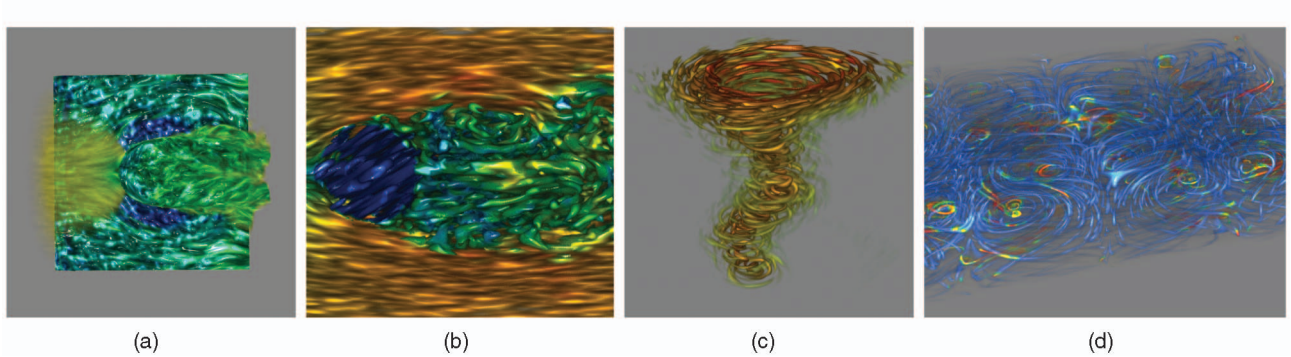

Fig. 9. Examples. (a) Semitransparent visualization of a large eddy simulation used for performance benchmarks. (b) Almost opaque rendering of the large eddy simulation. (c) Velocity masking for a tornado data set. (d) Benard convection with λ_2 vortex regions highlighted in yellow to red.

7 Results

We conducted tests of rendering performance on a Linux PC with an AMD x2 4400+ (2.2 GHz) CPU, 2 Gbytes of RAM, and a single NVIDIA 8800 GTX (768 Mbytes) GPU. All of the following performance numbers were collected for the large eddy data set shown in Fig. 9a on a 512^2 viewport with 20 LIC integration steps. The data set was stored in RGBA 8-bit format. We performed measurements regarding the texture format of the data set. The rendering was slowed down by approximately 3 percent when the 16-bit floating-point format was used and 17 percent with the 32-bit floating-point format. Each slab of our layered depth structure covered 50 samples for each ray. This number was chosen so that the instruction limit of fragment programs is not exceeded on older GPUs. Because of the chosen volume sampling distance not more than seven layers were needed to traverse the volume.

7.1 Rendering Performance

We compare two different transfer functions for the importance function: a rather semitransparent one (as displayed in Fig. 9a) and an almost completely opaque one. Fig. 8a compares different rendering options for the same camera that is used in Fig. 9a. Fig. 8a shows that nonoptimized rendering (i.e., no early abortion of unnecessary LIC computations) leads to a rendering speed that is mostly independent of the transfer function. The optimized versions lead to a significant increase in frame rate. For the opaque transfer function, early-ray termination alone is sufficient for a tremendous speedup. For the partly transparent transfer function, both optimization methods need to be combined for best results. The overall lower frame rate for the semitransparent case is caused by two factors. First, more volume samples contribute information to the final image, leading to a higher level of visible depth complexity. Second, the empty-space skipping implementations for both 3D texture slicing and ray casting are not optimal because the early-z test and the layered depth construction come at additional processing costs, and they do not always lead to an optimal skipping of fragment processing. Nevertheless, these optimization strategies lead to a substantial performance improvement compared with a brute-force computation and, thus, are effective to a large degree.

The total number of sample points for which the LIC integral is evaluated is shown in Table 1. In case of the almost opaque transfer function, only a small number of the overall volume samples contribute to the final image. When a semitransparent transfer function is applied, early-ray termination has much less effect, and only the combination with empty-space skipping leads to an efficient computation.

TABLE 1
Number of Calculated Sample Points
Using Different Rendering Options

Opaque transfer function

rendering option	ray casting	slicing
non-optimized	24,311,784	24,177,922
early-ray termination	786,425	524,287
early-ray + empty space	786,425	524,287

Semitransparent transfer function

rendering option	ray casting	slicing
non-optimized	23,571,332	23,457,528
early-ray termination	19,982,654	21,516,820
early-ray + empty space	3,137,511	2,966,550

TABLE 2
Impact of the Data Set Size on the Rendering Performance

Data set size	32^3	64^3	128^3	256^3	512^3
Render time [s]	0.90	0.93	0.98	1.12	2.05

7.2 Output Sensitivity

Fig. 8b compares the impact of the viewport size on rendering time. Here, the same scene was rendered with an increasing viewport size. When the viewport size is quadruplicated, e.g., from 512^2 to $1{,}024^2$, the rendering time increases only by a factor of 2.3. This points to the fact that the setup of the render passes plays a role and loses importance on larger viewports. The effect of the number of LIC samples is illustrated in Fig. 8c. The shown linear behavior implies a constant time per LIC sample C_{LIC}, as proposed in Section 3.2. In Table 2, the rendering times are shown for various data set sizes. The measurements were taken with resampled versions of the tornado data set. The transfer function was chosen in Fig. 3d. The output size on the image plane was the same for all measurements. Table 2 shows almost constant rendering times except for the slowdown between 256^3 and 512^3 textures, which can be explained by presumably more cache misses.

7.3 Varying Viewing Positions

Fig. 8d documents the performance for varying camera positions. Here, only the fully optimized rendering methods (early-ray termination with empty-space skipping) are used, with the same rendering parameters as before. Quasi-random camera positions (from a Halton sequence) are chosen uniformly on a sphere that surrounds the center point of the data. The camera always points toward the center of the data set. Fig. 8d summarizes the statistics for 120 camera positions using a box plot to show the median, the upper and lower quartiles, and the extreme frame rates.

7.4 Visual Results

Fig. 9 shows example images generated by 3D LIC. Figs. 9a and 9b visualize a large eddy simulation of a flow behind a cylindrical obstacle (data set size 256×128^2); semitransparent rendering is applied in Fig. 9a, whereas Fig. 9b uses a different viewpoint together with a clipping plane to show the interior of the flow. The tornado in Fig. 9c (here, for size 256^3) is extracted by an importance function that emphasizes high velocity magnitude. Another example of a feature criterion is used in the Benard convection (size $256 \times 64 \times 32$) in Fig. 9d: vortex extraction by λ_2. This image provides an example of focus-and-context visualization, with the focus (the vortices) in yellow to red and the surrounding flow in blue. The introductory image (Fig. 1) shows another example of feature-based focus-and-context 3D LIC, extracting vortex regions of another vortex-governed fluid flow (size $135 \times 225 \times 129$). Fig. 1 also illustrates the effect of adaptive rendering for its two camera views. Fig. 12 compares normalized and non-normalized vector field lengths to indicate the speed of flow. The upstream and downstream directions are visualized by streaks given by employing an asymmetric LIC filter kernel. Four time steps of an unsteady flow are shown in Fig. 10. The frames were depicted from a time-dependent tornado data set with 500 frames. As an example of a relatively large data set, we used the tornado data set with a resolution of 512^3. The data was stored in 16-bit floating-point RGBA format (normalized direction and magnitude), yielding 1 Gbyte of texture memory. We used the same CPU as above and an NVIDIA Quadro FX 5600 (1,536 Mbytes) to display this data set. We added different amounts of filtered Gaussian-distributed white noise to the data set. This emphasizes the capability of our approach to extract fine structures of large data sets. The first row of Fig. 11 contains the cross sections of the tornado with increasing noise. In the second row, the camera was positioned closer to the cross section (camera dolly). Due to the constant spatial resolution in the image plane finer details are visible. More examples, especially electronic videos, can be found in the supplemental material (http://doi.ieeecomputersociety.org/10.1109/TVCG.2008.25), and on our Web page.[3]

8 Discussion and Comparison with Previous Approaches

A comprehensive comparison with previous 3D flow visualization methods is difficult because of the large number of different previous techniques. Therefore, this section targets a comparison that focuses on the characteristics of our proposed visualization approach. First, we start with a high-level comparison between texture-based flow visualization and traditional particle-tracing methods that build explicit line geometry for characteristic lines. We refer to the first class as texture-based methods and to the latter as geometric methods. In the second half of this section, we focus our discussion on texture-based methods, comparing our approach with previous texture-based methods.

8.1 Comparison: Geometric and Texture-Based Methods

The main difference between geometric and texture-based methods is the way they represent lines or linelike structures. Geometric methods use explicit 3D positions along lines; positions are modified when traversing lines. In contrast, texture-based methods use an indirect representation by means of some kind of particle density function; the particle density function is given with respect to a spatially fixed domain, and a change of particle position is only indirectly represented by relative changes of the density function on neighboring spatial positions. Please note that we do not specify a certain type of discretization of the density function at this point but defer the discussion of texture representation to a later point. Using an analogy from fluid dynamics, geometric methods resemble a Lagrangian approach, whereas texture-based methods resemble an Eulerian approach.

The separation on this abstract level is beneficial because it allows us to clearly see the commonalities and differences between geometric and texture-based methods. Both approaches share fundamental characteristics and challenges of 3D flow visualization: spatial perception, shape and orientation perception, possible occlusion, appropriate seeding for lines, and extraction and highlighting of flow features. Therefore, basic strategies for improving perception, including illumination models (e.g., codimension-2 models),

3. http://www.vis.uni-stuttgart.de/texflowvis.

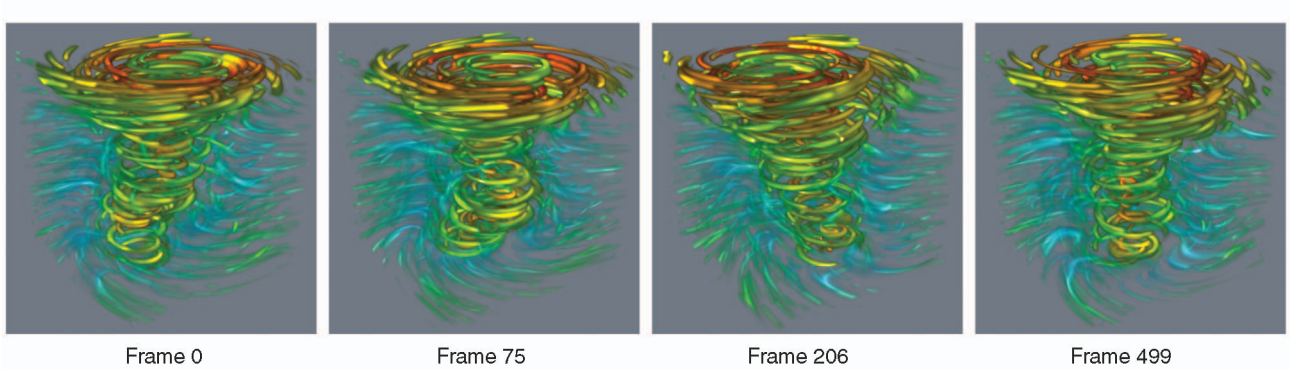

Fig. 10. Unsteady flow visualization: four time steps from an animation of a time-dependent tornado data set (500 frames in total).

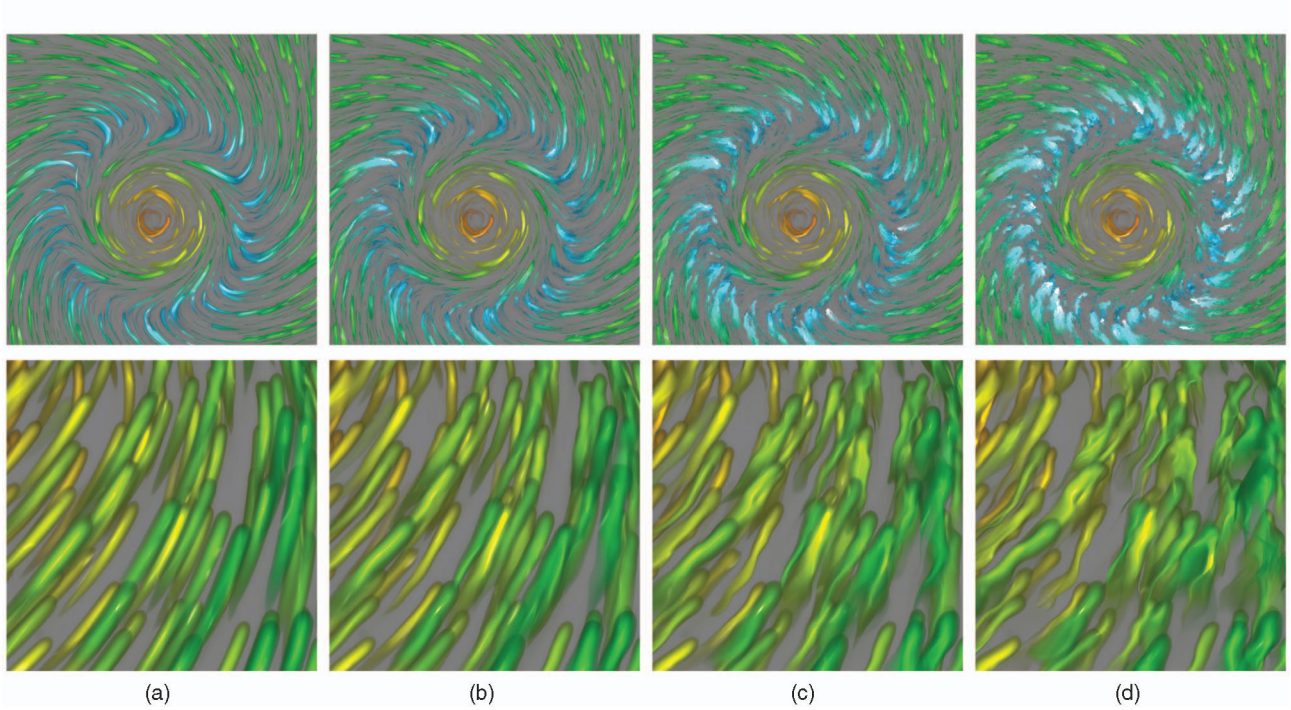

Fig. 11. Tornado (data set size 512^3) with added Gaussian-distributed white noise (increasing noise from left to right). The top row shows the complete cross section; in the bottom row, the camera was moved toward the data set to enlarge the structures.

line halos, and appropriate choice of line colors, can be shared. Similarly, strategies for feature extraction and space-variant seeding such as vortex detection, shock detection, flow topology, or interactive feature specifications can be adopted for both visualization strategies. In contrast, the major difference is the way how lines or linelike structures are represented. Geometric methods can easily represent thin long lines, and they are most efficient when only a few lines are present. Texture-based methods employ some kind of sampling of the density function and, thus, are capable of representing different styles of linelike structures: thin, thick, or even fuzzy "lines" that do not have a clear separation between a line and background. For example, the flexibility of fuzzy "lines" could be useful for uncertainty visualization that avoids sharp visual representations. In addition, texture-based methods are efficient at representing a large number of lines, they provide implicit spatial sorting, which is good for semitransparent rendering, and they allow for fast halo rendering by limb darkening.

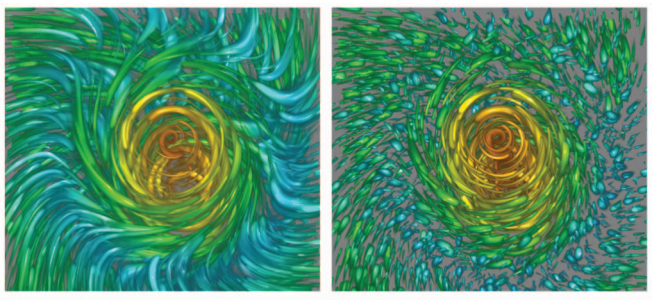

Fig. 12. Indicating the speed of flow. (a) Normalized vector field length. (b) Non-normalized vector field length. An asymmetric LIC filter kernel was used to depict the flow direction.

So far, the popularity of geometric methods is certainly rooted in their better rendering performance and ability to render thinner, clearer lines, compared to previous texture-based methods. The LIC method of this paper improves texture-based flow visualization in terms of rendering

performance and clear line rendering. In this way, we reduce the performance penalty of 3D LIC, while retaining the fundamental advantages of texture-based visualization. As discussed below, our 3D LIC method scales better than typical geometric methods for large data sets; therefore, there is some indication that 3D LIC might be particularly preferable for large-data visualization. On the other hand, geometric methods scale with the number of geometric primitives and, thus, are more suited for sparser representations, with long streamlines, or on rather large displays. Finally, perceptual investigations [43] show that streamlines of finite thickness combined with illumination are effective in improving the perception of spatial orientation; therefore, the moderately thin linelike structures of our 3D LIC method are appropriate from a perceptual point of view.

8.2 Comparison: Texture-Based Methods and 3D Line Integral Convolution

Previous texture-based methods, regardless of whether they require precomputation of the 3D texture or not, essentially adopt an object-space point of view: they first construct a 3D volume of the flow representation and then perform rendering. In contrast, we apply an image-space oriented sampling approach that has been successful in many other areas of computer graphics and visualization: "only compute what can be seen!" For example, view dependency has been proven successful for the simpler case of 2.5D texture-based flow visualization [24], [25], [27], [41], [47]. Another prominent example is ray tracing for surface rendering, which tries to determine the rendering computations mainly by image size and, therefore, scales better for large scenes than surface rasterization [42]. While this paradigm has already been proven advantageous for 3D LIC today, it will gain further advantage in the future because the size of 3D flow volume data sets is growing faster than the available image resolution. In particular, our adaptive view-dependent rendering allows users to zoom into large data sets that might not be completely displayable on the available screen space. This intrinsic view dependency of the computational load and the good scalability are not only preferable when compared with previous texture-based methods, but they might also provide a simpler approach to large-data visualization than geometric methods. In addition, our approach leads to intrinsically parallel computations (e.g., for volume rendering and LIC integration), coherent read access to memory (i.e., to the vector field and noise texture), and almost no write access to memory (only to the image plane). In contrast, previous texture-based methods require substantial write access to the density texture; similarly, geometric methods need to write information on streamlines. Therefore, the 3D LIC method of this paper is well suited for current and upcoming hardware architectures that show a trend toward a high level of parallelism (e.g., multicore CPUs) and an increasing gap between instruction and memory access speed.

Besides performance benefits, our technique offers a low-memory footprint because it avoids object-space data for the LIC density. Therefore, memory is available to store larger data sets. Similarly, the spatial frequency of the noise model can be adjusted without adding memory footprint, which allows us to render thin line structures. In addition, our technique recomputes all information for each frame and, thus, allows for time-varying streamlines in time-dependent vector fields. The only extra cost is the transfer of the varying vector fields to GPU memory.

Compared with 3D texture advection, which is the only other approach that is capable of a fully on-the-fly computation of dense 3D flow visualization, our visualization technique provides a significantly improved quality: LIC allows for arbitrary convolution filters that are superior to the exponential filter of texture advection, and the Lagrangian particle integration of LIC avoids the numerical diffusion and image blurring of the repeated resampling during semi-Lagrangian texture advection (see the related discussion [45] on quality). By employing an asymmetric filter kernel according to OLIC, we can show upstream and downstream direction similar to texture advection. The 3D streaks that result from asymmetric filtering resemble the 2D streaks that were used early on by Halley [12]: elongated streaks with their thick rounded ends pointing toward the flow direction. This strategy of thickening the streak toward its head is also recommended by Bertin [4] and backed up by the user study by Fowler and Ware [10]. Another user study by Laidlaw et al. [21], which targets a task-oriented comparison of different flow visualization techniques, also happens to show a generally good performance for techniques that use streamlets with broadened heads (techniques labeled "OSTR" and "GSTR" in their paper). Asymmetric filtering for 3D LIC leads to similar streak shapes on the image plane and, therefore, is appropriate for displaying upstream and downstream direction.

Finally, although there exist previous techniques for high-quality dense 3D flow visualization, they require some degree of precomputation, typically for particle tracing or the LIC volume. Therefore, these techniques are less suited for interactive exploration with varying camera parameters or for time-dependent data.

9 Conclusion and Future Work

We have introduced a largely output-sensitive visualization method for 3D LIC. Its key elements are 1) lazy evaluation of the LIC integral, 2) adaptive and view-dependent rendering through a variable LIC integration length and MIPmapping for 3D noise, and 3) efficient culling of volume samples that do not contribute to the final image by means of early-ray termination and empty-space skipping. The built-in adaptivity of our LIC method makes it generically applicable, leading to a rendering speed that is automatically adapted to the complexity of the visual representation, e.g., the screen resolution and the length of LIC streaks. We believe that this paradigm of view-dependent computations is highly advantageous, following the idea: "only compute what can be seen!" Temporal coherence under camera or object motion is guaranteed by modeling the noise as an object-aligned 3D texture. The perception of the shape and orientation of LIC streaks is facilitated by including line-based and gradient-based illumination alike. In particular, we have introduced an efficient gradient-based method that relies on precomputed noise gradients. We have also demonstrated that our visualization strategy lends itself to a fast GPU implementation that allows for the interactive exploration of 3D flow. In addition, feature-based visualization is supported by a generic importance function,

including a focus-and-context rendering of the highlighted features (focus) and the surrounding flow field (context).

Compared with previous fully interactive visualization by 3D texture advection, our approach provides a substantially improved quality due to Lagrangian integration, flexible filter design, and adjustable spatial detail. Although there exist previous techniques for high-quality dense 3D flow visualization, these techniques require precomputation, for example, for particle tracing or the LIC volume. Therefore, our approach is first to combine an interactive on-the-fly computation with high visualization quality. In addition, our technique offers a low-memory footprint because it avoids object-space data for the LIC density. Furthermore, the recomputation of the whole image for each frame allows the efficient visualization of unsteady flow by time-varying streamlines: the only extra cost is the transfer of the varying vector fields to GPU memory. We have presented the visualization of multiple time steps of an unsteady flow with streamlines.

In future work, other visualization metaphors of unsteady flow could be investigated such as pathlines or streaklines. In addition, the efficiency of GPU-based empty-space skipping could be improved by examining other methods known from GPU-based volume rendering such as brick-based culling of empty regions. Furthermore, specific techniques for large-data visualization could be included, for example, scaling the rendering performance by parallelization on GPU clusters or employing out-of-core methods and data streaming from disk. Finally, perceptual studies could clarify the effectiveness of 3D LIC techniques and what parameters and illumination models are appropriate for good shape and orientation perception, while adopting methods and strategies from [21] and [43].

ACKNOWLEDGMENTS

The tornado data set was kindly provided by Roger Crawfis (Ohio State University), the vortex flow (Fig. 1) by Ulrich Rist (Universität Stuttgart), and the large eddy simulation (Figs. 9a and 9b) by Octavian Frederich (TU Berlin). Special thanks to Bettina A. Weiskopf for proof reading.

REFERENCES

[1] A. Bair, D.H. House, and C. Ware, "Texturing of Layered Surfaces for Optimal Viewing," *IEEE Trans. Visualization and Computer Graphics,* vol. 12, no. 5, pp. 1125-1132, Sept./Oct. 2006.

[2] D.C. Banks, "Illumination in Diverse Codimensions," *Proc. ACM SIGGRAPH '94,* pp. 327-334, 1994.

[3] S. Bergner, T. Möller, D. Weiskopf, and D.J. Muraki, "A Spectral Analysis of Function Composition and Its Implications for Sampling in Direct Volume Visualization," *IEEE Trans. Visualization and Computer Graphics,* Proc. IEEE Visualization, vol. 12, no. 5, pp. 1353-1360, Sept./Oct. 2006.

[4] J. Bertin, *Semiology of Graphics.* Univ. of Wisconsin Press, 1983.

[5] B. Cabral, N. Cam, and J. Foran, "Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware," *Proc. Symp. Volume Visualization,* pp. 91-98, 1994.

[6] B. Cabral and L.C. Leedom, "Imaging Vector Fields Using Line Integral Convolution," *Proc. ACM SIGGRAPH '93,* pp. 263-270, 1993.

[7] R. Cranley and T. Patterson, "Randomization of Number Theoretic Methods for Multiple Integration," *SIAM J. Numerical Analysis,* vol. 13, pp. 904-914, 1976.

[8] K. Engel, M. Hadwiger, J.M. Kniss, C. Rezk-Salama, and D. Weiskopf, *Real-Time Volume Graphics.* A.K. Peters, 2006.

[9] C. Everitt, "Interactive Order-Independent Transparency," white paper, *Nvidia,* 2001.

[10] D. Fowler and C. Ware, "Strokes for Representing Univariate Vector Field Maps," *Proc. Graphics Interface,* pp. 249-253, 1989.

[11] J. Gallier, *Geometric Methods and Applications: For Computer Science and Engineering.* Springer, 2001.

[12] E. Halley, "An Historical Account of the Trade Winds, and Monsoons, observable in the Seas between and near the Tropicks, with an attempt to Assign the Phisical Cause of the said Winds," *Philosophical Trans.,* vol. 16, pp. 153-168, 1686/1692.

[13] A. Helgeland and O. Andreassen, "Visualization of Vector Fields Using Seed LIC and Volume Rendering," *IEEE Trans. Visualization and Computer Graphics,* vol. 10, no. 6, pp. 673-682, Nov./Dec. 2004.

[14] A. Helgeland and T. Elboth, "High-Quality and Interactive Animations of 3D Time-Varying Vector Fields," *IEEE Trans. Visualization and Computer Graphics,* vol. 12, no. 6, pp. 1535-1546, Nov./Dec. 2006.

[15] V. Interrante, "Illustrating Surface Shape in Volume Data via Principal Direction-Driven 3D Line Integral Convolution," *Proc. ACM SIGGRAPH '97,* pp. 109-116, 1997.

[16] V. Interrante and C. Grosch, "Strategies for Effectively Visualizing 3D Flow with Volume LIC," *Proc. IEEE Visualization,* pp. 421-424, 1997.

[17] J. Jeong and F. Hussain, "On the Identification of a Vortex," *J. Fluid Mechanics,* vol. 285, pp. 69-94, 1995.

[18] A. Kaufman and K. Mueller, "Overview of Volume Rendering," *The Visualization Handbook,* C.D. Hansen and C.R. Johnson, eds., pp. 127-174, Elsevier, 2005.

[19] M.-H. Kiu and D.C. Banks, "Multi-Frequency Noise for LIC," *Proc. IEEE Visualization,* pp. 121-126, 1996.

[20] J. Krüger and R. Westermann, "Acceleration Techniques for GPU-Based Volume Rendering," *Proc. IEEE Visualization,* pp. 287-292, 2003.

[21] D.H. Laidlaw, R.M. Kirby, C.D. Jackson, J.S. Davidson, T.S. Miller, M. da Silva, W.H. Warren, and M.J. Tarr, "Comparing 2D Vector Field Visualization Methods: A User Study," *IEEE Trans. Visualization and Computer Graphics,* vol. 11, no. 1, pp. 59-70, Jan./Feb. 2005.

[22] R.S. Laramee, C. Garth, J. Schneider, and H. Hauser, "Texture Advection on Stream Surfaces: A Novel Hybrid Visualization Applied to CFD Simulation Results," *Proc. EG/IEEE VGTC Symp. Visualization (Eurovis '06),* pp. 155-162, 2006.

[23] R.S. Laramee, H. Hauser, H. Doleisch, B. Vrolijk, F.H. Post, and D. Weiskopf, "The State of the Art in Flow Visualization: Dense and Texture-Based Techniques," *Computer Graphics Forum,* vol. 23, no. 2, pp. 143-161, 2004.

[24] R.S. Laramee, B. Jobard, and H. Hauser, "Image Space Based Visualization of Unsteady Flow on Surfaces," *Proc. IEEE Visualization,* pp. 131-138, 2003.

[25] R.S. Laramee, J.J. van Wijk, B. Jobard, and H. Hauser, "ISA and IBFVS: Image Space Based Visualization of Flow on Surfaces," *IEEE Trans. Visualization and Computer Graphics,* vol. 10, no. 6, pp. 637-648, Nov./Dec. 2004.

[26] G.-S. Li, X. Tricoche, and C. Hansen, "GPUFLIC: Interactive and Accurate Dense Visualization of Unsteady Flows," *Proc. EG/IEEE VGTC Symp. Visualization (Eurovis '06),* pp. 29-34, 2006.

[27] L. Li and H.-W. Shen, "View-Dependent Multi-Resolutional Flow Texture Advection," *Proc. SPIE Conf. Visualization and Data Analysis (VDA),* 2006.

[28] O. Mallo, R. Peikert, C. Sigg, and F. Sadlo, "Illuminated Lines Revisited," *Proc. IEEE Visualization,* pp. 19-26, 2005.

[29] N. Max, "Optical Models for Direct Volume Rendering," *IEEE Trans. Visualization and Computer Graphics,* vol. 1, no. 2, 99-108, June 1995.

[30] H. Niederreiter, *Random Number Generation and Quasi-Monte Carlo Methods.* SIAM, 1992.

[31] F.H. Post, B. Vrolijk, H. Hauser, R.S. Laramee, and H. Doleisch, "The State of the Art in Flow Visualization: Feature Extraction and Tracking," *Computer Graphics Forum,* vol. 22, no. 4, pp. 775-792, 2003.

[32] C. Rezk-Salama, P. Hastreiter, C. Teitzel, and T. Ertl, "Interactive Exploration of Volume Line Integral Convolution Based on 3D-Texture Mapping," *Proc. IEEE Visualization,* pp. 233-240, 1999.

[33] S. Röttger, S. Guthe, D. Weiskopf, T. Ertl, and W. Straßer, "Smart Hardware-Accelerated Volume Rendering," *Proc. EG/IEEE TCVG Symp. Visualization,* pp. 231-238, 2003.

[34] G. Schussman and K.-L. Ma, "Anisotropic Volume Rendering for Extremely Dense, Thin Line Data," *Proc. IEEE Visualization,* pp. 107-114, 2004.
[35] H.-W. Shen, G.-S. Li, and U.D. Bordoloi, "Interactive Visualization of Three-Dimensional Vector Fields with Flexible Appearance Control," *IEEE Trans. Visualization and Computer Graphics,* vol. 10, no. 4, pp. 434-445, July/Aug. 2004.
[36] D. Stalling and H.-C. Hege, "Fast and Resolution Independent Line Integral Convolution," *Proc. ACM SIGGRAPH '95,* pp. 249-256, 1995.
[37] S. Stegmaier, M. Strengert, T. Klein, and T. Ertl, "A Simple and Flexible Volume Rendering Framework for Graphics-Hardware-Based Raycasting," *Proc. EG/IEEE VGTC Workshop Volume Graphics,* pp. 187-195, 2005.
[38] Y. Suzuki, I. Fujishiro, L. Chen, and H. Nakamura, "Case Study: Hardware-Accelerated Selective LIC Volume Rendering," *Proc. IEEE Visualization,* pp. 485-488, 2002.
[39] A. Telea and J.J. van Wijk, "3D IBFV: Hardware-Accelerated 3D Flow Visualization," *Proc. IEEE Visualization,* pp. 233-240, 2003.
[40] J.J. van Wijk, "Image Based Flow Visualization," *ACM Trans. Graphics, Proc. ACM SIGGRAPH '02,* vol. 21, no. 3, pp. 745-754, 2002.
[41] J.J. van Wijk, "Image Based Flow Visualization for Curved Surfaces," *Proc. IEEE Visualization,* pp. 123-130, 2003.
[42] I. Wald, A. Dietrich, and P. Slusallek, "An Interactive Out-of-Core Rendering Framework for Visualizing Massively Complex Models," *Proc. Eurographics Symp. Rendering,* pp. 81-92, 2004.
[43] C. Ware, "3D Contour Perception for Flow Visualization," *Proc. Symp. Applied Perception in Graphics and Visualization (APGV '06),* pp. 101-106, 2006.
[44] R. Wegenkittl, E. Gröller, and W. Purgathofer, "Animating Flow Fields: Rendering of Oriented Line Integral Convolution," *Proc. Computer Animation,* pp. 15-21, 1997.
[45] D. Weiskopf, "Iterative Twofold Line Integral Convolution for Texture-Based Vector Field Visualization," *Math. Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration,* T. Möller, B. Hamann, and R. Russell, eds., Springer, http://www.visus.uni-stuttgart.de/~weiskopf/publications, 2008.
[46] D. Weiskopf, *GPU-Based Interactive Visualization Techniques.* Springer, 2006.
[47] D. Weiskopf and T. Ertl, "A Hybrid Physical/Device-Space Approach for Spatio-Temporally Coherent Interactive Texture Advection on Curved Surfaces," *Proc. Graphics Interface,* pp. 263-270, 2004.
[48] 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," *Proc. Workshop Vision, Modeling, and Visualization (VMV '01),* pp. 439-446, 2001.
[49] D. Weiskopf, T. Schafhitzel, and T. Ertl, "Real-Time Advection and Volumetric Illumination for the Visualization of 3D Unsteady Flow," *Proc. EG/IEEE VGTC Symp. Visualization (Eurovis '05),* pp. 13-20, 2005.
[50] D. Weiskopf, T. Schafhitzel, and T. Ertl, "Texture-Based Visualization of Unsteady 3D Flow by Real-Time Advection and Volumetric Illumination," *IEEE Trans. Visualization and Computer Graphics,* vol. 13, no. 3, pp. 569-582, May/June 2007.
[51] A. Wenger, D.F. Keefe, S. Zhang, and D.H. Laidlaw, "Interactive Volume Rendering of Thin Thread Structures within Multivalued Scientific Data Sets," *IEEE Trans. Visualization and Computer Graphics,* vol. 10, no. 6, pp. 664-672, Nov./Dec. 2004.
[52] M. Zöckler, D. Stalling, and H.-C. Hege, "Interactive Visualization of 3D-Vector Fields Using Illuminated Stream Lines," *Proc. IEEE Visualization,* pp. 107-113, 1996.

Martin Falk received a diploma degree in computer science from the Universität Stuttgart, Germany, in 2007. He is currently working toward the PhD degree at the Visualization Research Center, Universität Stuttgart (VISUS). His research interests include flow visualization, GPU methods, and computer graphics. He is a student member of the ACM and the IEEE.

Daniel Weiskopf received the MSc (Diplom) degree in physics and the PhD degree in physics from Eberhard-Karls-Universität Tübingen, Germany, and the Habilitation degree in computer science from the Universität Stuttgart, Germany. From 2005 to 2007, he 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 a member of the ACM SIGGRAPH, the Gesellschaft für Informatik, and the IEEE Computer Society.