Flow Charts: Visualization of Vector Fields on Arbitrary Surfaces

Guo-Shi Li, Xavier Tricoche, *Member*, *IEEE*, Daniel Weiskopf, *Member*, *IEEE Computer Society*, and Charles Hansen, *Senior Member*, *IEEE*

Abstract—We introduce a novel flow visualization method called *Flow Charts*, which uses a texture atlas approach for the visualization of flows defined over curved surfaces. In this scheme, the surface and its associated flow are segmented into overlapping patches, which are then parameterized and packed in the texture domain. This scheme allows accurate particle advection across multiple charts in the texture domain, providing a flexible framework that supports various flow visualization techniques. The use of surface parameterization enables flow visualization techniques requiring the global view of the surface over long time spans, such as Unsteady Flow LIC (UFLIC), particle-based Unsteady Flow Advection Convolution (UFAC), or dye advection. It also prevents visual artifacts normally associated with view-dependent methods. Represented as textures, *Flow Charts* can be naturally integrated into hardware accelerated flow visualization techniques for interactive performance.

Index Terms—Flow visualization, textures, graphics hardware.

1 Introduction

A visual assessment of 3D transient flow phenomena is essential in a broad range of scientific, engineering, and medical applications. In many cases, the analysis of a 3D vector field can be reduced to the investigation of the 2D structures produced by its interaction with the boundary of the object under consideration. Typical examples of such analysis for fluid flows include airfoils in aeronautics, engine wall and exhaust pipes in the automotive industry, and rotor blades in turbo machinery. Other applications in biomedicine focus on the interplay between bioelectric fields and the surface of an organ. In each case, numerical simulations of increasing size and sophistication are becoming instrumental in helping scientists and engineers reach a deeper understanding of the flow properties that are relevant to their task. The scientific visualization community has concentrated a significant research effort on the design of visualization methods that convey local and global structures occurring at various spatial and temporal scales in flow simulations. In particular, emphasis has been put on the interactivity of the corresponding visual analysis, identified as a critical aspect for the effectiveness of the proposed algorithms.

- *G.-S. Li is with the School of Computing and the Scientific Computing and Imaging Institute, University of Utah, 72 South Central Campus Drive, 3750 WEB, Salt Lake City, UT 84102. E-mail: lig@sci.utah.edu.*
- *X. Tricoche is with the Department of Computer Science, Purdue University, 305 N. University Street, West Lafayette, IN 47907. E-mail: xmt@purdue.edu.*
- *D. Weiskopf is with the Visualization Research Center and Visualization and Interactive Systems Institute, Universität Stuttgart, Nobelstrasse 15, 70569 Stuttgart, Germany. E-mail: weiskopf@vis.uni-stuttgart.de.*
- *C. Hansen is with the School of Computing, University of Utah, 50 South Central Campus Drive, 3154 MEB, Salt Lake City, UT 84102. E-mail: hansen@cs.utah.edu.*

Manuscript received 11 Oct. 2007; revised 28 Feb. 2008; accepted 17 Mar. 2008; published online 31 Mar. 2008.
Recommended for acceptance by A. Pang.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-2007-10-0160.
Digital Object Identifier no. 10.1109/TVCG.2008.58.

A recent trend within flow visualization research focuses on image space methods [17], [34]. This framework offers compelling means to tackle the computational complexity of visualization techniques supporting flows defined over curved surfaces. The key feature of this approach lies in its ability to efficiently produce a dense texture representation of the flow without explicitly computing a surface parameterization. This is achieved by projecting the flow on the visible part of the surface onto the image plane, where subsequent texture generation is performed in the image space through backward integration and iterative blending. Although the use of partial surface parameterization obtained by projection results in an impressive performance gain, a limitation of this setting is that texture patterns stretching beyond the visible part of the self-occluded surface become incoherent due to the lack of global representation of the surface flow. Popping artifacts are present when the surface is rotated as partial parameterizations are in general not fully consistent across consecutive frames. This limitation also restricts the use of dye advection to investigate nonlocal features in the vector field. Fig. 1 illustrates the visualization of a synthetic spiral helix flow defined on the surface of a cylinder using dense texture combined with dye advection. Only with the global view of the flow can the dye material correctly circumnavigate the surface to convey the nonlocal structure. Hence, to extend existing planar texture-based methods to flows defined on curved surfaces in a general way, a global representation of the surface flow in the 2D texture domain is needed.

One way to attain such a representation is surface parameterization, which provides the mapping between the texture domain and the physical space. This technique has been successfully utilized in many problems involving curved surfaces, such as texture mapping, texture synthesis, or mesh processing. Although many surface parameterization methods exist, their use in dense texture flow visualization has been limited so far. The specific requirements imposed by flow visualization methods partially explains

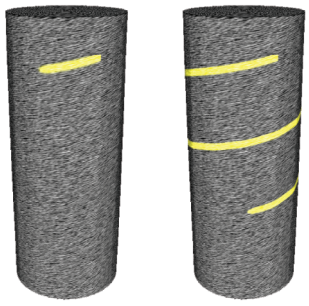


Fig. 1. Visualizing the spiral helix flow on a cylinder using dye advection combined with dense texture.

this disconnect. Many dense texture flow visualization methods are based on the Lagrangian principle of particle advection, which mimics the physical process of massless particles carried away by the flow into which they are released. To accommodate a wide range of surfaces, cuts are typically necessary in surface parameterization to convert the associated flows to the texture domain while reducing distortion. This inevitably introduces discontinuity in the global flow representation, causing many numerical schemes used for particle advection to fail. Moreover, the ability to redirect particles across discontinuities in the texture space is necessary to ensure artifact-free visualizations. Finally, interactivity is an essential dimension of any effective visual data exploration. Therefore, the aforementioned problems require algorithmic solutions that fully leverage the parallel computational structure of modern Graphics Processing Unit (GPU) architectures.

To address these problems, we propose a novel framework designed to support a wide range of dense texture visualizations for flows over arbitrary curved surfaces using existing surface parameterization schemes. Our scheme, called *Flow Charts*, segments the surface into patches that are eventually converted to charts to provide a global view of surface flow in the texture domain. In order to efficiently enable particle advection across discontinuities on chart boundaries in the texture space, the patches are extended to overlap their neighbors in the physical space. The extended region provides each local chart with a smooth representation of its direct vicinity in the flow as well as with the interchart adjacency information, both required for accurate and nondisrupted particle advection. The vector field and the patch adjacency relation are naturally represented as textures, enabling efficient GPU implementation of state-of-the-art dense texture flow visualization algorithms requiring the global view of the flow field, such as GPUFLIC [21], Unsteady Flow Advection Convolution (UFAC) [39], and level-set dye advection [35].

In summary, the specific contributions of this paper are

- introduction of a generic framework providing the global view of flows on curved surfaces using state-of-the-art surface parameterization algorithms;
- a set of schemes and data structures enabling particle advection with flow fields with discontinuities in texture space suitable for GPU implementation;
- adaptation of planar dense texture flow visualization algorithms to curved surfaces.

The contents of this paper are organized as follows: Related work in dense texture flow visualization is discussed in Section 2. Section 3 elaborates on the concept of *Flow Charts* and details their computation. Section 4 demonstrates the versatility of our algorithm by showing examples of GPUFLIC, UFAC, and level-set dye advection on curved surfaces using *Flow Charts*. Implementation details are covered in Section 5. Section 6 provides discussion and results. Finally, we conclude this paper by summarizing our contributions and pointing out promising avenues for future research in Section 7.

2 Related Work

Flow visualization using dense textures is an active research field. Since the seminal works by van Wijk [32] and Cabral and Leedom [4], many methods have been proposed (such as [13], [14], [33], and [39]). Laramee et al. [16] provided a comprehensive survey on this rich subject. Weiskopf et al. [37] proposed a mathematical framework to analyze various techniques in terms of spatial and temporal coherence and later investigated visual quality issues from a signal processing perspective [36].

Despite the tremendous efforts devoted to 2D flows, existing methods addressing flows defined over surfaces are relatively few. Forssell and Cohen [9] generated the texture representation for such a flow using Line Integral Convolution (LIC) in the parameter space and then applied it to the surface for visualization. This method was later improved by Mao et al. [24], who used a noise texture of multiple frequencies to compensate visual quality issues caused by distortion of the mapping. For an arbitrary curved geometry, Mao et al. [25] computed LIC using a procedurally defined solid noise texture encompassing the entire surface. LIC computation in 3D was only carried out on texels intersecting with the visible part of the surface, which are determined using ray casting. Another method by Battke et al. [3] computed LIC textures individually on every triangle of the surface. The triangle adjacency was explicitly maintained for establishing patterns across triangle boundaries. Due to the complexity and the technologies available at the time, these methods are designed as offline processes.

The rapid evolution of graphics hardware in recent years had inspired a new generation of algorithms leveraging the resulting computational power. Lately, two similar GPU-based methods, Image Space Advection (ISA) [17] and Image-Based Flow Visualization for Curved Surfaces (IBFVS) [34], were proposed to generate dense textures for unsteady flow defined on curved surfaces. Called image space methods, the flow on the visible part of the surface is projected onto the image plane that is then used for texture generation. A comparison and analysis of these two methods can be found in [18]. Image space methods can achieve interactive frame rates since the entire pipeline can be carried out on the GPU. Due to its view-dependent nature, however, frame-to-frame coherency in texture patterns is not guaranteed. Weiskopf and Ertl [38] addressed this problem with a dual-domain approach. Using a noise texture defined in 3D, the particle advection is conducted back and forth between the image and the physical space to ensure the consistency of texture patterns and to avoid the "edge-crossing" problem, which

otherwise needs to be specially dealt with in image space methods. The frequency of the noise texture is dynamically adapted to mitigate aliasing in the resulting texture patterns.

3 Flow Charts Visualization Framework

The primary goal of this research is to devise a general framework supporting a wide range of texture-based flow visualization algorithm on polygonal surfaces. Surface parameterization provides a global view of the flow in the texture space. For comprehensive surveys on surface parameterization research, see [7] and [8]. Given an appropriate surface parameterization, any flow visualization methods designed for 2D flows can be used to generate dense flow textures, which are then texture-mapped onto the surface in the physical space for visualization. In this framework, the visualization techniques are orthogonal to the choice of surface parameterization method, which is determined by various criteria such as error threshold and computation cost. However, there are two main challenges to this approach. In the following, we first explain these key issues before introducing our Flow Charts visualization framework.

Flow on surface. The first issue in our framework involves accurately expressing physical flows tangential to the surface in the texture space. The tangential flow is obtained by projecting the 3D vectors attached on the surface to local tangent planes. The surface description in our work is a polygonal mesh corresponding to the computation grids of Computational Fluid Dynamics (CFD) simulations, as typical in practice. A surface parameterization is needed to express tangential vectors in a globally consistent coordinate frame since the polygonal mesh is only C^0 continuous. Unlike Catmull-Clark subdivision surfaces used in [31], the parameterization of polygonal surfaces needs to be computed explicitly. To cope with a wide range of surfaces without excessive distortion, we opt for the texture atlas scheme by Maillot et al. [23] rather than single-chart method such as Geometry Image [11]. In our approach, the surface is segmented into patches that are then parameterized and packed into the 2D texture space. Following the standard naming scheme, the footprint of a patch in the texture space is referred to as a chart, providing a mapping to the corresponding surface region in the physical space. The flow defined on the surface can then be transformed to the texture space using the surface parameterization. We will come back to this issue with further details in Section 3.3.

Particle advection in texture space. With a texture atlas approach in mind, we now consider particle advection across multiple charts, which is the fundamental building block in many texture-based flow visualization methods. Consider a time-varying vector field $\mathbf{v}(t, \mathbf{x})$ that assigns a vector to each position $\mathbf{x}$ in space D at time t. From the Lagrangian perspective, the trajectory of a massless particle $\mathbf{x}_{\mathbf{path}}$ induced by $\mathbf{v}$ can be formulated as an ordinary differential equation:

$$\frac{\mathrm{d}\mathbf{x}_{\mathbf{path}}(t; \mathbf{x_0}, t_0)}{\mathrm{d}t} = \mathbf{v}\big(t, \mathbf{x}_{\mathbf{path}}(t; \mathbf{x_0}, t_0)\big), \tag{1}$$

with the initial condition $\mathbf{x}_{\mathbf{path}}(t_0) = \mathbf{x_0}$. In order to guarantee the existence and uniqueness of the solution, the vector field $\mathbf{v}$ needs to satisfy the Lipschitz condition:

$$|\mathbf{v}(\mathbf{x}_1) - \mathbf{v}(\mathbf{x}_2)| \leq K|\mathbf{x}_1 - \mathbf{x}_2|, \tag{2}$$

with a positive constant K for all $\mathbf{x}_1, \mathbf{x}_2$ in D. This constraint is challenging for the aforementioned texture atlas approach since the flow in the vicinity of chart boundaries in the texture space is discontinuous due to the cuts. A common strategy to deal with a similar situation is to replicate boundary pixels, such as in [5] and [19]. In these methods, only a small number of pixels (usually one or two) adjacent to the boundary are involved in the computation. This solution is, however, not suitable in our setting. Indeed, in particle advection, the positions between updates are not necessarily in adjacent pixel locations. Furthermore, once a particle exits a chart during advection, it needs to be transported to the corresponding adjacent chart to maintain a smooth trajectory in the physical space. Battke et al. [3] addressed this issue by explicitly maintaining and checking triangle adjacency of the entire mesh using an AVL search tree, which is challenging for parallel GPU hardware implementation.

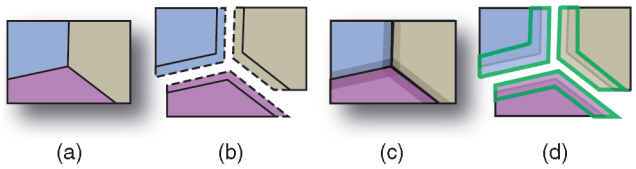


Fig. 2. Two-dimensional illustration of constructing overlapping patches. (a) The surface is initially segmented into disjoint regions, each constitutes the center zone of a final patch. (b) Attaching buffer zones (in dashed lines) to disjoint regions in (a). (c) Patches now overlap with their neighbors. (d) Overlapping regions are highlighted with green silhouettes.

To address these issues, we propose a mesh segmentation scheme suitable for flow visualization in which each patch is initially disjoint and then overlapped with adjacent ones by a small extent. We refer to the formerly disjoint patch region as the "center zone," while the extended region is called the "buffer zone." The buffer zone serves two purposes. First, after parameterization in the texture space, it provides a continuous representation of the flow close to the chart's center zone boundary by including the neighboring flow information. Second, it encodes the adjacency relationship between abutting patches so that particle advection can be correctly performed across multiple charts in the texture space. Due to the symmetry by construction, part of the center zone of a given patch is also overlapped with the buffer zone of its neighboring patches. This part of the center zone together with the buffer zone of the patch is collectively called the "overlapping region" (see Fig. 2). It is used to capture the texture patterns generated across the boundary between the center zone and the buffer zone of a chart to guarantee artifact-free visualizations in the physical space. The overlapping patches, along with associated flow and adjacency relation, are parameterized and packed to the texture space. These data are converted to a set of textures called *Flow Charts* to facilitate efficient GPU implementation. With *Flow Charts* and additional schemes to perform particle advection across multiple charts and to

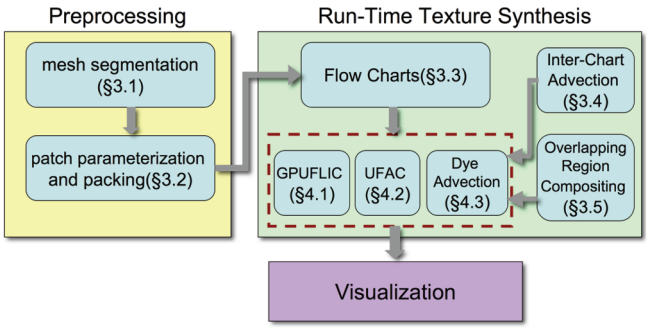


Fig. 3. Texture-based visualization framework using Flow Charts. The numbers refer to the sections that describe each step.

ensure the smoothness of texture patterns across chart boundaries, a wide range of dense texture flow visualization algorithms can be readily extended to visualize flows on arbitrary curved surfaces. Fig. 3 provides an overview of our framework.

3.1 Mesh Segmentation and Parameterization

The first stage in our framework requires segmenting and parameterizing the surface into overlapping patches. This is achieved by first creating disjoint patches from a set of seed triangles, which are then extended by a small region surrounding their boundaries. Since the main purpose of these patches is to transform the flow to the texture space, care must be taken that the mesh is properly segmented and the distortion induced by parameterization is minimized. Additionally, due to the limited amount of texture memory on the graphics card, the patch configuration resulted from segmentation must achieve high packing efficiency. With these concerns in mind, we propose an automatic mesh segmentation and parameterization algorithm based on the principle of region growing inspired by [5]. It consists of the following steps:

1. Find candidate seed triangles.
2. Grow disjoint patches iteratively with shape and distortion control.
3. Attach buffer zone and compute patch parameterization.

In the first step of our method, a set of triangles from the mesh is selected as the candidate seed triangles for region growing by using an adaptive surface sampling scheme by Meyer et al. [26]. In this method, a set of samples is distributed on the surface, each representing a locally flat region. In a series of region growing processes, a set of disjoint regions is formed one by one from *some* of these candidate seed triangles. This is different from [5], in which seed triangles are found by analyzing mesh features (as further discussed in Section 3.1.1) and each seed corresponds to one final patch. During the region growing process, the quality of parameterization is tested on the fly to ensure low distortion. Other criteria, such as shape and size of the region, are also taken into consideration for better packing efficiency. Once the mesh is fully covered, these disjoint regions constitute the center zones of the final patches. A small region of adjacent triangles is then attached to each patch to form the buffer zone. Last, the parameterization of these overlapping patches is computed. In the rest of this section, we discuss each of these aspects in more detail.

3.1.1 Finding Candidate Seeds

The selection of seed triangles is an essential issue for automatic mesh segmentation schemes. Some methods, such as D-charts [15] and Rectangular Multi-Charts Geometry Images [5], take the iterative Lloyd-Max approach [22] where the number and the location of seeds are repeatedly computed in a converging process. Although this approach is conceptually simple and can deal with a wide range of surfaces, it is computationally expensive for large CFD meshes. Another way to tackle this problem is to "reverse engineer" the location of seeds based on the criteria of desirable resulting patches. Flat and compact patches are favorable as they yield better results in parameterization and packing. Therefore, it is preferable that patch boundaries are aligned with sharp features while the patch centers should be away from them. Based on this principle, one can explicitly estimate feature curves on the mesh, from which flat locations ideal for seeds can be determined such as in [20]. This requires properly estimating mesh features. In our experiments, however, we found that CFD meshes used in practical applications do not lend themselves to simple local feature detection techniques found in the graphics and mesh processing literature. The mesh tessellation in this type of data can be highly nonuniform, as it is adaptively refined according to the local importance of physical phenomenon in the numerical simulation. This is atypical to many graphics schemes, in which isotropic tessellation is usually assumed. Remeshing or smoothing is not an option because it inevitably alters the original vector data. To overcome these difficulties, we propose a scheme to determine a series of candidate seed triangles without direct mesh processing using the adaptive sampling scheme for implicit surfaces by Meyer et al. [26]. First, the polygonal mesh is converted to an implicit form by scan conversion. It is then sampled to a user-specified number of points. The sample points are confined on the implicit surface with sampling rates proportional to local curvatures. These samples are then projected back to the original mesh to pinpoint the closest intersecting triangles as the candidate seeds. The sampling density can be seen as a rough estimate of local flatness of the mesh. Based on this measure, each candidate seed triangle is given a priority for region growing.

3.1.2 Iterative Patch Construction

Each candidate seed triangle s_T obtained in Section 3.1.1 is associated with the average distance P from the corresponding sample s_T to its N closest neighbors s_n on the implicit surface C. That is,

$$P(s_T) = \frac{\Sigma_N |C(s_T) - C(s_n)|}{N}. \quad (3)$$

This value describes the size of the locally flat neighborhood it represents and is used as the priority value of s_T in the priority queue for region growing. A spatial subdivision data structure is used to track the neighborhood information of sample points s_i. For details, see [26].

Starting from the seed triangle with the largest locally flat neighborhood, the patch is iteratively constructed by

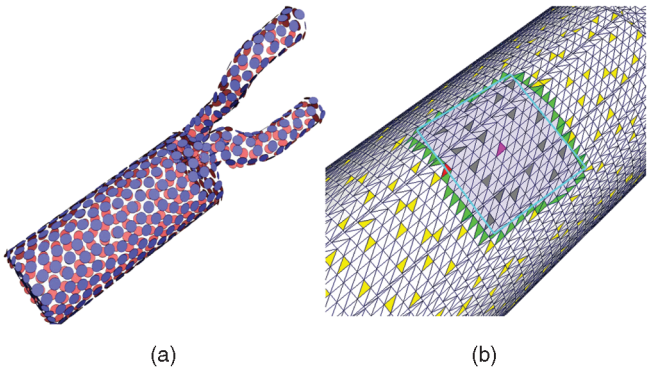


Fig. 4. Iterative patch construction with candidate seed triangles. (a) Candidate seed triangles are found by adaptive sampling of the implicit representation of the polygonal mesh. (b) Region growing with shape control. The current patch (light gray) is initially grown from a seed triangle (pink). Seed triangles invalidated during the region growing process, active seed triangles, and triangles bordering the current patch are shown in dark gray, yellow, and green, respectively. Among the bordering triangles, the one with the minimal L^∞ distance (red) is computed with the estimated local frame in the texture space (cyan).

including faces adjacent to its boundary edges. We maintain a list of patch ownership flags for every triangle of the mesh. At this point, each triangle can only belong to exactly one patch. During the construction process, unused candidate seed triangles can be included into the patch and become unavailable. We ensure the growing region is a topological disk by checking if it contains holes. The process continues until certain criteria halt the region expansion. When this happens, the next available candidate seed triangle in the priority queue becomes the new active seed for region growing. If there are multiple candidate seed triangles with the same priority, the one farthest away from the centroid of the latest constructed patch is chosen. The geodesic distance is approximated by iteratively propagating the one-ring patch boundary. This ensures that we always start constructing patches from a flat region rather than a sharp area. When all candidate seed triangles are exhausted, any unallotted faces are assigned to the closest patches to fully partition the mesh into disjoint regions (see Fig. 4).

3.1.3 Patch Growing Control

Several criteria are considered to regulate the patch growing process. First, patches should not span across sharp edges on the mesh because the tangential flow there is likely to be discontinuous. Otherwise, this would lead to two flow regions of conflicting directions that coexist on the same chart, causing ambiguous patterns in the resulting texture. To avoid this problem, during the patch growing process, we check the difference between the normal vector of the prospective triangle to be included and that of its neighbor already on the patch. It is rejected from consideration if this value is larger than a user-specified threshold. Optionally, the average normal vector of the patch can be maintained to test against the normal vector of the candidate. The candidate is rejected if the angle difference is larger than a user-specified threshold. This is to ensure that the resulting patch is relatively flat to minimize potential errors in parameterization.

The mesh segmentation has a tremendous impact on the distortion induced by parameterization. Since in our framework the mesh is segmented by iterative patch generation, the parameterization test is integrated with the region growing process to ensure the mesh is segmented into a set of low distortion patches. We use Least Squares Conformal Maps by Levy et al. [20] to compute patch parameterization and L^2 stretch error [28] as error measure. If the induced L^2 stretch error is larger than a user-specified threshold, the current patch is halted from further growth and a new candidate seed is selected to form the next patch.

The last control criterion is concerned with packing efficiency. To effectively utilize the texture space, it is desirable that the shape of the patches be approximately rectangular in the texture space. We adapt the shape control mechanism described in [5] for this purpose. In this method, rectangular patches are attained by favoring the triangle of the closest L^∞ distance in the local tangent space to be included in the patch during the growing process. The L^∞ distance between two points α and β is defined as

$$L^\infty(\alpha, \beta) = \max_i(|\alpha_i - \beta_i|), \tag{4}$$

where i is the dimension index. Note that equidistant lines defined with L^∞ are rectangles. In our experiments, we found that principal component analysis (PCA) is not suitable for obtaining the local frame to compute the L^∞ distance. This is because the vertex distribution on CFD meshes can be highly nonuniform due to local refinements for simulation purposes. To solve this problem, following [10], we first compute the convex hull of the patch vertices projected onto their linear least square fitting plane and then find the minimal area bounding box of the convex hull using the rotating caliper algorithm by Eppstein [6]. The eigenvectors of the bounding box are thus used as the local frame for computing the L^∞ distance (see Fig. 4).

3.1.4 Buffer Zone Attachment and Overlapping Region

Patches obtained in Section 3.1.2 are mutually exclusive and constitute the center zone of the final patches. They are attached to buffer zones by a similar growth procedure but with a tighter angle difference threshold than the one used in Section 3.1.3. The "thickness" of the buffer zone, d_B, needs to be large enough such that the outgoing trajectories of particles originated from the center zone can always be captured by the buffer zone of the corresponding patch. Assuming Runge-Kutta fourth order (RK4) is the chosen numerical scheme, $\mathbf{y}_*$ is the particle position, h is the maximum allowable step size, and $\mathbf{v}$ is the vector field, the particle displacement in one iteration is

$$\mathbf{y_{n+1}} - \mathbf{y_n} = \frac{h}{6}(\mathbf{k_1} + 2\mathbf{k_2} + 2\mathbf{k_3} + \mathbf{k_4}), \tag{5}$$

where

$$\begin{aligned}
\mathbf{k_1} &= \mathbf{v}(t_n, \mathbf{y_n}),\\
\mathbf{k_2} &= \mathbf{v}\left(t_n + \frac{h}{2}, \mathbf{y_n} + \frac{h}{2}\mathbf{k_1}\right),\\
\mathbf{k_3} &= \mathbf{v}\left(t_n + \frac{h}{2}, \mathbf{y_n} + \frac{h}{2}\mathbf{k_2}\right),\\
\mathbf{k_4} &= \mathbf{v}(t_n + h, \mathbf{y_n} + h\mathbf{k_3}).
\end{aligned}$$

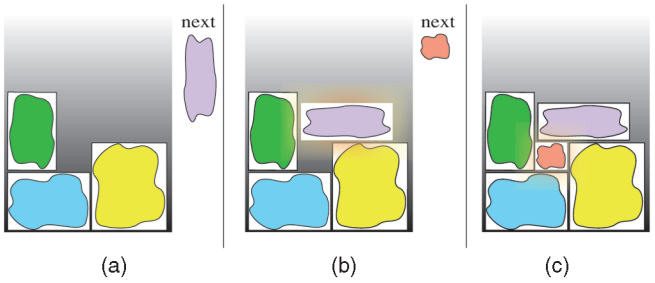


Fig. 5. Chart packing. (a) The currently packed charts with their bounding boxes and an incoming chart to be inserted (marked by "next"), which in (b) is rotated by 90 degrees to minimize the increase in height. (c) A chart falling through the blockade in the existing packing.

Suppose v_{max} is the maximum vector magnitude throughout the time sequence, by definition $\|\mathbf{k_i}\| \leq v_{max}$, $i = 1 \ldots 4$, thus (5) leads to

$$d_B \geq h \cdot v_{max}\left(\frac{1}{6}+\frac{1}{3}+\frac{1}{3}+\frac{1}{6}\right). \quad (6)$$

When a triangle T is included in the buffer zone of patch P, it is by construction covered by multiple patches since it must be in the center zone of another patch Q adjacent to P. This creates the overlapping region. For each triangle, we maintain a list of ownership flags to keep track of the overlapping patch configuration. They are used in creating the redirection map (Section 3.4) and overlapping region compositing (Section 3.5).

3.2 Packing

After the mesh is segmented and parameterized, individual charts are packed to a uniform texture space to facilitate subsequent texture computation. Since the texture memory is typically limited, it is important to tightly pack the charts to minimize unused texture space. Several heuristic algorithms have been proposed since the packing problem is known to be NP-hard [27]. Igarashi and Cosgrove [12] suggested to sort the chart bounding boxes by height and then divided the sorted sequence into zigzag bands, which are used as the order of inserting charts into the texture space. Levy et al. [20] proposed a packing scheme similar to the popular video game "Tetris." In this method, a horizontal curve bounding all currently inserted charts is maintained. It is tested against the lower boundary of the next chart to be inserted to determine the best placing position, which minimizes the wasted texture space. This method was later improved by Sander et al. [29] in which each incoming chart is considered for 16 possible orientations.

Inspired by the aforementioned schemes, we propose a "Tetris"-like packing scheme balancing the ease of implementation and packing efficiency (see Fig. 5). Since the charts are forced to be approximately rectangular by construction, we use the chart bounding box for packing and only consider two orientations (horizontal or vertical). The width of the texture space for packing is determined by the square root of the total chart area times a scaling coefficient. Larger charts are inserted before smaller ones as they are relatively less flexible for filling up fractional space. The placement and orientation of the incoming chart are chosen to minimize the peak height of the packing. Unlike conventional Tetris, in our method, the incoming chart can fall through existing blocking charts if it can be fit into an available space underneath. In this case, the peak height of the packing is not changed. This test can be easily done since bounding boxes instead of actual charts are used for packing. After all charts are inserted, their parameterization coordinates are rescaled.

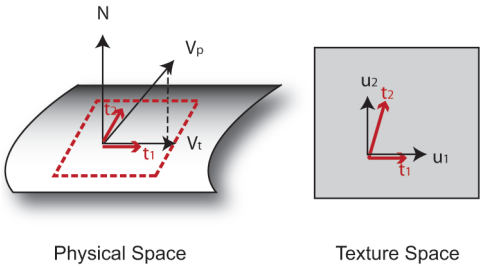


Fig. 6. Transforming physical space vector V_p to the texture space. The dashed quadrilateral shows the local tangent plane.

3.3 Flow Charts

The parameterization, flow data, and adjacency relation associated with every chart are converted into *Flow Chart* textures using rasterization. There are three types of textures: the *parameterization map*, *flow map*, and *redirection map*. The parameterization map encodes the surface parameterization and is used to create the flow map. The flow map represents the flow in the texture space, the essential ingredient in flow visualization. The redirection map encodes the chart adjacency relation, which is essential for interchart particle advection. It redirects particles across multiple patches, as we will further discuss in Section 3.4. All three maps are in the same resolution, which is determined by scaling the packing result in Section 3.2 by a constant. For a given surface flow data set, there are exactly one parameterization map and one redirection map. Each time step corresponds to a unique flow map. Flow charts of a given data set are orthogonal to the choice of visualization technique and are computed in a preprocessing step.

Parameterization map and flow map. The parameterization map $\mathbb{P}$ is created by rendering each patch using the texture space coordinates $\mathbf{u} = (s, t)$ defined at every vertex as the position attribute, and the associated physical space coordinates $\mathbf{x_p} = (x, y, z)$ as the color attribute. It is then used to create the flow map $\mathbb{F}$, which is constructed in the following steps (see Fig. 6). First, the physical space vector $\mathbf{v_p} = (v_x, v_y, v_z)$ on each patch is rasterized into the texture space with $\mathbf{u}$ as the vertex position attribute and $\mathbf{v_p}$ as the color attribute. Next, $\mathbf{v_p}$ associated with each texel of $\mathbb{F}$ is projected to the local tangent plane spanned by basis $\beta_t = \{\mathbf{t_1}, \mathbf{t_2}\}$ and becomes a tangential vector $\mathbf{v_t}$:

$$\mathbf{v_t} = \mathbf{v_p} - (\mathbf{v_p} \cdot \mathbf{N})\mathbf{N}, \quad (7)$$

where $\mathbf{N} = \frac{\mathbf{t_1} \times \mathbf{t_2}}{\|\mathbf{t_1} \times \mathbf{t_2}\|}$ is the normal vector of the local tangent plane, and

$$\mathbf{t_1} = \frac{\partial \mathbb{IP}(\mathbf{u})}{\partial s}, \quad \mathbf{t_2} = \frac{\partial \mathbb{IP}(\mathbf{u})}{\partial t}. \quad (8)$$

Finally, the tangential vector $\mathbf{v_t}$ in the physical space is transformed to the texture space by coordinate frame

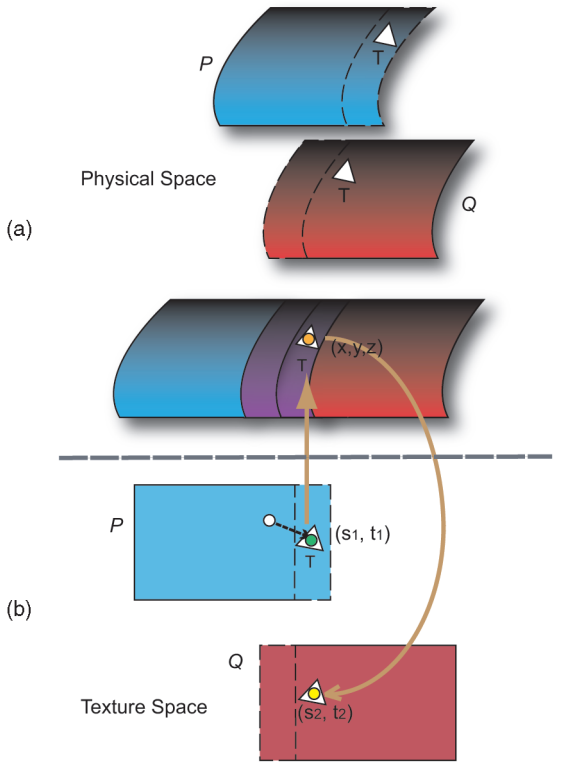


Fig. 7. Interchart particle advection. (a) Patch P (blue) and Q (red) with buffer zones in dashed lines individually shown and in overlapping configuration. (b) A particle entering buffer zone of chart P redirected to the corresponding location on Q.

transformation from β_t to β_u, where $\beta_u = \{\mathbf{u_1}, \mathbf{u_2}\}$ is the basis of the texture space and

$$\mathbf{u_1} = \frac{\mathbf{t_1}}{\|\mathbf{t_1}\|}, \quad \mathbf{u_2} = \frac{\mathbf{N}}{\|\mathbf{N}\|} \times \frac{\mathbf{t_1}}{\|\mathbf{t_1}\|}. \tag{9}$$

Redirection map. The patch adjacency relation can be understood as follows: patch P and Q are adjacent if there exists a triangle that simultaneously resides in the center zone of P and the buffer zone of Q, or in the buffer zone of P and the center zone of Q. In the redirection map, we denote the patch adjacency relation in the texture space with position correspondence in terms of triangle ID. Beside the gaps between charts, every pixel in the redirection map carries a flag to indicate whether it is in patch center zone and a unique triangle ID in the physical space that it belongs to. The creation of the redirection map is similar to that of the parameterization map except that the color attribute is replaced by the center/buffer zone indicator flag and a triangle ID.

3.4 Interchart Particle Advection

Particle advection is a basic computational aspect of most flow visualization techniques. In the context of our method, the ability to perform particle advection across chart boundaries is essential to generate flow patterns in the texture space that yield visually coherent results in the physical space. Flow Charts support this operation in the following way. Particles are always seeded in the center zone of individual charts in the texture space. The additional flow information provided in the buffer zones

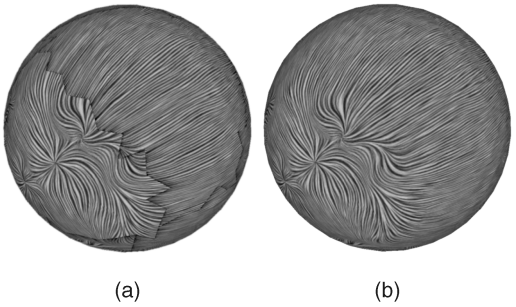


Fig. 8. (a) Artifacts in texture patterns caused by inconsistent partial particle traces in overlapping regions. (b) Fixed with overlapping region compositing.

enables the use of higher order numerical integration schemes like fourth-order Runge-Kutta for good accuracy. In each iteration of the advection process, the particle position is first updated by the numerical scheme, which is then used to sample the redirection map to determine the texture position at which the next iteration of integration will be resumed at. If the position reached after one integration step lies within the center zone of the same chart, no redirection is necessary and the particle remains at its current position. If, on the other hand, the particle has left the center zone and entered the buffer zone of the same chart, it is redirected to the center zone of another chart by the use of the redirection map. The redirection process is illustrated in Fig. 7. At particle position (s_0, t_0) in the buffer zone of patch P, the matching physical space coordinates (x, y, z) are obtained by sampling the parameterization map $\mathcal{P}$ at (s_0, t_0). From the redirection map $\mathcal{R}$, the triangle ID T at (s_0, t_0) is first used to obtain the spatial coordinates of the corresponding three vertices to compute the barycentric coordinates of (x, y, z) in T. This is then used to interpolate another set of texture space coordinates of vertices of T with respect to Q to compute the position on Q, where a particle will be redirected to.

3.5 Compositing Overlapping Regions

Although the particle redirection scheme described in Section 3.4 promises smooth transition among patches, it alone is not enough to achieve artifact-free visualization. Most dense texture methods rely on particle traces, rather than discrete points of particle locations, to produce textures for visualization. When a particle enters the buffer zone of patch P and is then redirected to the adjacent patch Q, its trace made on P right before the transition is not visible to Q. In general, the traces that connect the successive positions of each particle must be consistently replicated to all corresponding positions in the texture space to ensure pattern coherence in the physical space. Fig. 8 illustrates the artifacts if this consistency is not preserved. Since only the overlapping region of a chart is covered by multiple patches, only particle traces in this area must be duplicated onto another adjacent chart in the physical space. Fig. 9 illustrates this situation. In Fig. 9a, a particle moves from u_0 to u_{1P} on patch P after one advection iteration. u_0 and u_{1P} are in the center zone and the buffer zone of P, respectively, and u_b is the intersection

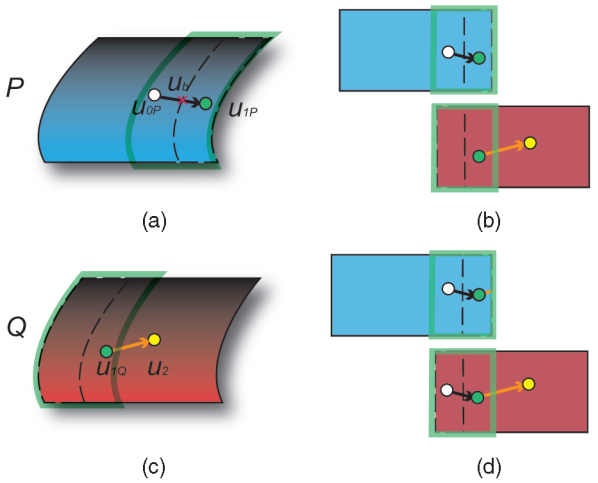


Fig. 9. Compositing overlapping regions. Buffer zones are shown with dashed silhouettes and overlapping regions are marked with green boundaries. (a) and (b) A particle enters buffer zone of patch P, being redirected to patch Q, and continues on patch Q. (c) Transit particle traces generated in (a) and (b) on corresponding charts. (d) Transit paths on chart P and Q symmetrically composited on each other.

point of $\overline{u_0u_{1P}}$ and the center/buffer zone boundary. In Fig. 9b, the particle is transported to u_{1Q} on the adjacent patch Q and is further advected to u_2. Although the particle position is transported correctly due to the redirection scheme, the transit trace $\overline{u_0u_{1p}}$ is not. Specifically, the partial segment $\overline{u_0u_b}$ should be replicated onto the buffer zone and $\overline{u_bu_{1P}}$ onto the center zone of Q. Symmetrically, $\overline{u_{1Q}u_2}$ also shall be replicated to P to ensure consistency. Note that for simplicity the particle in Fig. 9 starts in the overlapping region of P. If this were not the case, only the partial trace in the overlapping region would need to be composited.

The compositing operation is performed with the triangle ownership flags described in Section 3.1.4. Using the ownership flags and multiple parameterization coordinates with respect to all patches that a triangle belongs to, every transit particle trace on the triangles in an overlapping region is replicated to all corresponding areas in the texture space. Consider an overlapping region triangle T with parameterization coordinates (s_i^P, t_i^P), $i = 1 \ldots 3$ with respect to chart P. It also carries another set of coordinates (s_i^Q, t_i^Q), $i = 1 \ldots 3$ with respect to chart Q. To transport all transit particle traces on T from P to Q, T is composited upon Q with (s_i^P, t_i^P) as texture coordinates and (s_i^Q, t_i^Q) as vertex positions. The composition from Q to P is achieved by inverting the position and texture coordinates. It is possible that more than two patches cover the overlapping region simultaneously. In this case, the symmetric composition is done in multiple passes. Implementation details are provided in Section 5.

4 Dense Texture Flow Visualization with Flow Charts

With Flow Charts, a wide range of dense texture flow visualization algorithms can be carried out in the texture space, allowing efficient GPU-based implementation. Particularly, visualization techniques requiring the global view of the flow can be correctly generated without view-dependent artifacts. In this section, we describe how to adapt GPUFLIC, particle-based UFAC, and level-set dye advection to arbitrary curved geometries using our framework.

4.1 GPUFLIC

GPUFLIC [21] is an iterative reformulation of the original Unsteady Flow LIC (UFLIC) [30] method, which efficiently generates a series of flow textures to visualize unsteady 2D flows using graphics hardware. In this method, particles are iteratively released at every pixel in the domain to sample the current flow texture, which is initialized as white noise. Each particle is then advected forward across the space-time domain and deposits the associated scalar value to pixels along its path. The values thus accumulated at each pixel are then averaged and filtered to create the next frame of the visualization. The particle life span is chosen so as to enhance the coherence between consecutive frames.

Using Flow Charts, GPUFLIC can be extended to generate flow textures on curved surfaces with the following adjustments. First, the noise texture in the texture space is obtained by using the parameterization map to sample a multifrequency noise texture defined in the physical space [38]. The use of a 3D noise texture ensures consistent noise across patches. The noise frequency varies locally according to the current viewing parameter of the surface in the physical space to address possible aliasing issues, as discussed in [38]. Second, particles are only released in central zones in the texture domain and are possibly redirected after each update of their positions, as described in Section 3.4. Third, partial traces made by particles crossing chart boundaries are composited upon correspondent regions on other charts using the method described in Section 3.5. Finally, high-pass filtering and noise jittering are performed on the texture generated in the computational space to enhance the contrast of resulting patterns.

4.2 Unsteady Flow Advection Convolution

UFAC [37] is a texture-based visualization scheme for unsteady 2D flows. When visualizing unsteady flows, the goals of temporal and spatial coherence are often conflicting. To address these issues, UFAC first creates the evolution of the noise texture induced by the flow and then performs LIC convolution on it to depict flow structures. The length of the convolution kernel is adjusted based on the unsteadiness of the flow at a given pixel location. The space-time continuum is constructed using backward advection. The performance of the temporal evolution is improved in the later version of UFAC [39], where forward particle tracing with radial basis functions is used to simulate the temporal propagation of a density field. A fixed number of particles are advected with randomized life spans initially adjusted by local divergence. When a particle expires, it is recycled and reseeded into a random position. In order to compensate the loss or gain of particle density induced by flow divergence, a probability-based scheme derived from the continuity equation is used to adjust particle life spans to ensure an even distribution of particles in the entire domain over time.

We extend the forward advection version of UFAC to curved surfaces using Flow Charts. The forward advection stage can be performed in the texture space using the

particle redirection scheme described in Section 3.4, similar to GPUFLIC. Likewise, the contributions from the radial basis functions of particles are captured and blended to the corresponding region in the texture space using the compositing scheme described in Section 3.5. In order to achieve a uniform particle distribution across charts, in addition to the probability-based scheme we also introduce an explicit removal mechanism to gradually remove particles from regions of high particle concentration. This is achieved by periodically checking the density of particles in the neighborhood of each pixel in the texture domain. When the particle density of the region a particle steps into is larger than the average number of particles in the ideal distribution, its life span is stochastically reduced to gradually decrease the particle density in the local area. Since patches overlap with their neighbors in the physical space, the density threshold for a given pixel location in the texture space needs to be divided by the number of patches it belongs to so that uniform distribution can be attained after compositing overlapping regions. After constructing the density field in the texture space using Flow Charts, LIC computation is conducted across multiple charts with the same scheme for particle redirection. Periodic histogram equalization is then applied to the resulting texture to boost image contrast.

4.3 Level-Set Dye Advection

Dye advection is a widely used experimental laboratory technique to explore and visualize nonlocal structures in the fluid medium, which is oftentimes transparent to human eyes. The foreign dye material is continuously injected into a fixed spot to be swept away by the flow, revealing the spatial and temporal structures originating from the injection site. In computer-generated flow visualizations, dye advection can be easily combined with noise-based dense representations (such as UFAC and GPUFLIC) to provide interactive and user-centric experiences to explore and understand flow structures. The traditional way to simulate dye advection is backward texture advection [13]. In this method, the properties of the dye (e.g., color) are represented as a texture. Since the Lagrangian formulation of particle advection is symmetric, the transport of dye material can be realized by integrating back in time at each pixel and then using the new pixel location to sample the property texture. Although it is easy to implement and is very efficient on modern graphics hardware, dye patterns generated by backward texture advection are usually blurry and decay rapidly due to numerical diffusion. Weiskopf [35] tackled this issue by using a level-set approach to confine dye material from being dispersed. Rather than colors in this method the texture represents a distance field, depicting the interface between the dye material and the background as a level set. The dispersion of the dye-background interface resulted from numerical diffusion, thus can be corrected by periodical level-set reinitialization.

The level-set dye advection can be easily extended to curved surfaces using Flow Charts. The dye injection site is specified by the user with a mouse click in the physical space. It is then translated to the texture space using the redirection map to initialize the level-set distance field. Note that the injection site in the physical space can correspond to multiple spots in the texture space if it is in an overlapping region. The distance field is transported in the texture space by backward advection. If the new pixel location resulted from backward advection falls in the buffer zone, it is redirected in the same manner as described in Section 3.4. The overlapping region compositing is used to ensure the level-set representation is continuous across chart boundaries. The resulting patterns can be overlaid with other dense texture presentations in the physical space for visualization.

5 Implementation

The proposed framework is implemented in C++, OpenGL, and Cg/CgFX on a standard Windows PC. We use GNU Triangulated Surface Library [2] and Computational Geometry Algorithms Library [1] for mesh segmentation and parameterization. After flow charts are converted to a set of textures, dense texture methods are run on the GPU. In the following, we discuss several aspects with regard to implementation details.

Mesh/patch data structures. Since patches overlap in the physical space, proper data structures are necessary to maintain the requisite information in flow charts. Every triangle of the mesh has a unique ID. Each patch maintains a list of the triangle IDs it covers. The mesh also keeps track of a list of pointers for each triangle to indicate which patch covers it. For each vertex it contains, a patch maintains a flag to indicate whether it is in the center zone or buffer zone, which triangle it belongs to, plus the texture space coordinates. When a triangle is included in the buffer zone of a patch, all other patches that also cover this triangle are also notified so that they can register that this triangle is now covered by other patches, thus allowing for the compositing of overlapping regions.

Two-stage compositing. The overlapping region composition described in Section 3.5 is a symmetric operation. It is done in two stages to avoid "double-dosing"; that is, patterns on triangle A copied to triangle B then repeatedly added back to A when B composites its patterns onto A. This problem is solved by two-stage compositing with an additional blank render target. In the first stage of composition, the original texture is read and the result is written to the blank render target. Next, the render target is composited back to the original texture as a whole.

Particle engine. All flow chart textures are stored on the GPU in the 32-bit floating point RGBA format. The particle advection is implemented by using two 32-bit floating point textures, `current` and `next`, that holds the successive particle positions between updates. The texture representing the `next` position is set to the render target and is updated using the `current` position along with flow charts as the input of a fragment shader implementing RK4 and particle redirection. The `current` and `next` texture are swapped after each iteration. Since the video memory is limited, for unsteady data sets, the physical space vector data is streamed to GPU during the course of the visualization to generate corresponding flow maps.

Position texture to graphics primitives. The particle positions stored in the textures need to be transformed into renderable primitives (i.e., line segments and point sprites

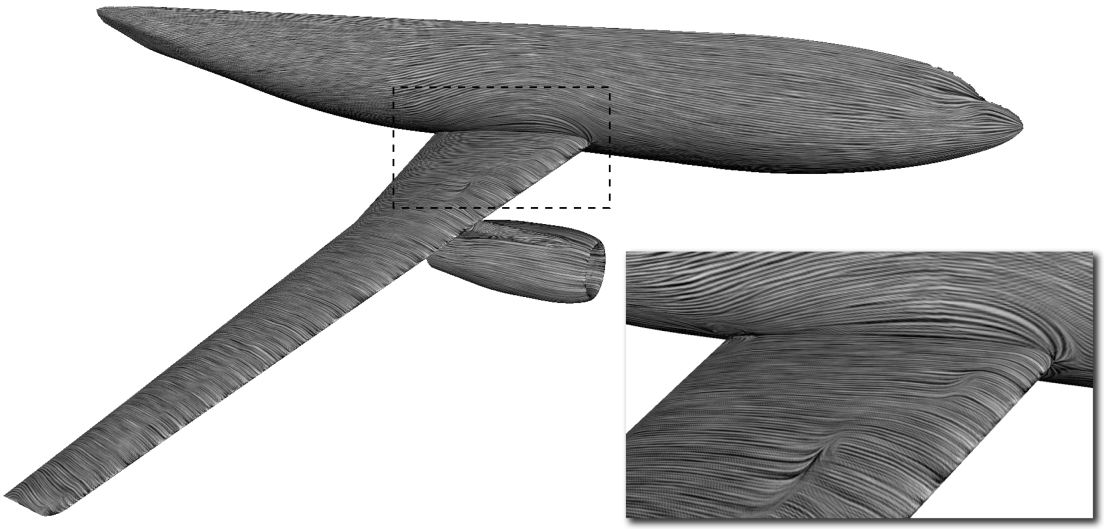


Fig. 10. Airliner data set visualized using GPUFLIC. The close-up image at the bottom right highlights flow features in the vicinity where the airfoil root and fuselage join.

in GPUFLIC and UFAC, respectively) to generate visualizations. In OpenGL, one solution is *copy-to-vertex buffer* via the combination of vertex/pixel/frame buffer object extensions. The particle positions in the texture can be directly copied to the storage space of vertex buffer objects on the graphics card, avoiding time-consuming CPU readback. Another possibility is to use vertex texture fetch (VTF) supported by some graphics hardware. In this scheme, a stream of vertex IDs is sent to GPU, which is used as texture coordinates to fetch the actual particle position from the texture with the vertex shader. Last, if the latest DirectX 10 generation graphics hardware is available, the transformation from computation results to graphics primitives is trivial using the Geometry Shader functionality.

Geometry and parameterization coordinates. In our framework, every vertex on the mesh carries multiple sets of coordinates, including potentially multiple pairs of texture space coordinates with respect to one or more patches it belongs to, plus the vertex position in the physical space. These coordinates are used for overlapping region compositing, visualizing resulting textures in the physical space, and particle redirection. In order to avoid CPU/GPU traffic and to save precious video memory, they are represented as vertex/pixel buffers with dynamic bindings so that only one copy of data is maintained.

6 Results and Discussion

We have conducted experiments with the Flow Charts framework on seven steady and unsteady flow data sets defined over curved surfaces using the dense texture techniques described in Section 4. In the case of fluid flow data sets, the viscous effects exhibited by the flow at the object boundary lead to what is known as a no-slip boundary condition. In other words, the flow velocity goes quickly to zero as one approaches the surface of the object. In that case, the surface flow considered in the proposed results is the shear stress vector field, which is derived from the first-order derivative of the 3D velocity. The associated patterns correspond to the so-called wall streamlines and are those observed in wind tunnel visualizations applying

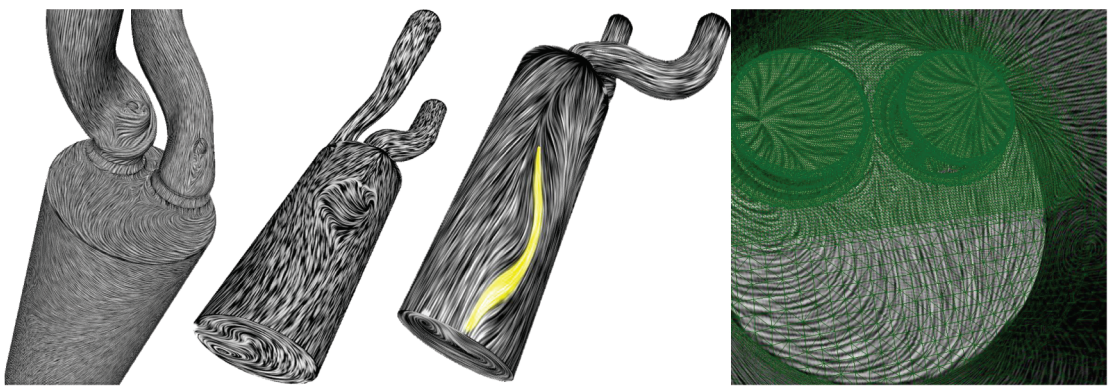
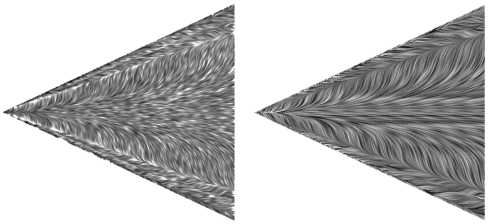


Fig. 11. Engine cylinder. (a) GPUFLIC result showing intake pipes connecting the combustion chamber, and openings of the piston (notice small bores on each pipe). (b) Swirling motions on the bottom and side of the combustion chamber depicted in UFAC. (c) A different view in UFAC, augmented with dye advection. (d) Pistons embedded inside the combustion chamber.



(a) (b)

Fig. 12. Side-by-side comparison of (a) UFAC and (b) GPUFLIC applied to the upper side of the delta wing.

oil films on the object body. The data sets corresponding to biomedical data sets show the nonzero restriction of the bioelectric field on the considered surface:

- `Cylinder` (Fig. 11) shows the swirl motion of a computer-simulated flow on the surface of an engine cylinder. The nontrivial geometry of this data set is composed of several components, including the combustion chamber (the main cylindrical body), two air intake pipes, and two corresponding pistons embedded inside the combustion chamber (see Fig. 11). In order to achieve optimal efficiency, the design goal is to achieve the ideal mixture of fresh air and vaporized fuel. In this case, the swirling motion spirals around an axis more or less aligned with the cylinder, which is a common design in diesel engines.
- `Delta wing` (Fig. 12) corresponds to a steady flow simulation of a sharp delta wing at low speed and high angle of attack. The key flow structures exhibited by this type of design in such critical flight configurations are systems of vortices that are located on both sides of the wing. The visualization of the shear stress vector field reveals several adjacent separation and attachment lines along the leading edges that correspond to those vortices. Their geometry, and in particular their curvature, is important in the study of the vortex breakdown phenomenon.
- `Airliner` (Fig. 10) shows shear stress flow on the surface of a commercial airliner. The simulation contains a steady-state flow around a standard plane configuration. Symmetry is leveraged for computational efficiency. The original 3D grid has 8.4 million cells.
- `ICE train` (Fig. 13) is the result of a simulation of a high-speed train traveling at a velocity of about 250 km/h with wind blowing from the side at an incidence angle of 30 degrees. The wind causes vortices to form on the lee side of the train, creating a drop in pressure that has adverse effects on the train's ability to stay on the track. These flow structures induce separation and attachment flow patterns on the train surface. They can be clearly seen close to the salient edges of the geometry.
- `Sphere` (Fig. 14) is a transient simulation of a turbulent flow past a spherical obstacle. The vortices created by the interaction of the flow with the surface produce complex separation and attachment patterns on the back side of the sphere. The visualization helps elucidate the temporal evolution of these structures along the course of the simulation.
- `Brain` (Fig. 15) was computed to study electroencephalogram (EEG) source localization. In this case, the forward problem (i.e., the impact of a known electric source activity on the bioelectric potential measured on the head surface) was solved over a high-resolution finite-element mesh. The nature of the bioelectric field implies that it remains constrained to the head surface. The patterns

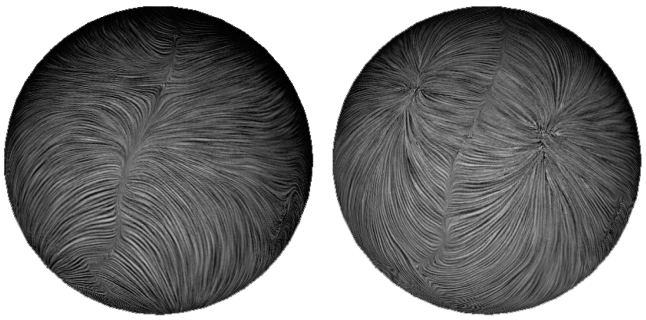
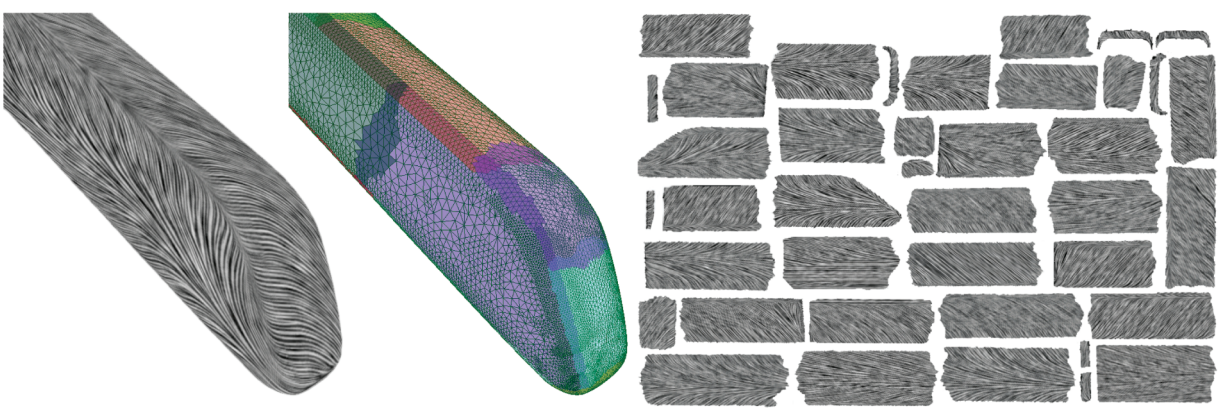
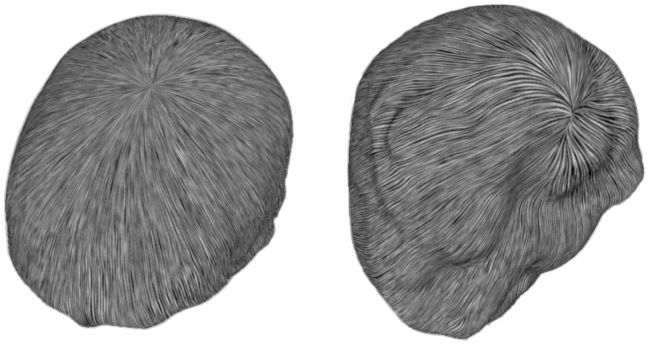


Fig. 14. Time-dependent sphere data set visualization in GPUFLIC.



(a) (b) (c)

Fig. 13. ICE train. (a) GPUFLIC result. (b) Patch configurations. (c) Chart packing.



(a) (b)

Fig. 15. (a) Top and (b) bottom views of the Brain data set.

exhibited by this vector field are a sink and a source whose locations relate to the position and orientation of the dipolar source, as well as to the conductivity properties of the underlying brain tissue.

- Heart (Fig. 16). The last data set shows the restriction of the bioelectric field to the epicardium, as computed using a finite-element simulation initialized with boundary conditions provided by experimentally acquired values of the electric potential at sparse locations on the heart surface. The context of this simulation is the study of ischemia. The visualization reveals the complex patterns exhibited by the bioelectric field and is of interest to investigate its relationship with the underlying heart anatomy. The image presents two views with slightly different viewing angles, showing consistent patterns in the physical space due to the use of surface parameterization.

Table 1 shows the basic statistics of the data sets used in our experiments and timing figures of surface segmentation/parameterization. Since these are computed only once before visualization in a preprocessing step, they do not affect the speed of the actual Flow Charts visualization. Table 2 provides the frame rates of runtime visualization using GPUFLIC and UFAC optionally with dye advection in different parameter settings. For GPUFLIC, we show the frame rates for different particle life spans (ttl). For UFAC, performance of particle advection, particle advection with LIC, and with additional histogram equalization are reported. The size of viewport is 800 by 800 in all

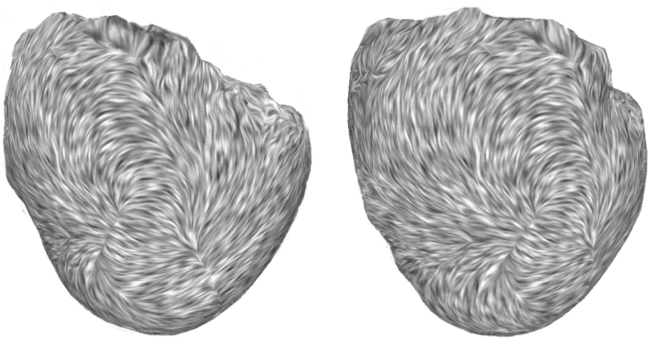


Fig. 16. Two views of the Heart data set visualization in UFAC with slightly different viewing angles.

TABLE 1
Timings (in Seconds) of Mesh Segmentation/Parameterization and Packing for Different Data Sets

Dataset	# triangle	# vertex	# patch	seg.& param.	packing
ICE	93886	46974	42	174.8△ /s	5.41 s
Delta Wing	63920	127836	22	212.3△ /s	2.83 s
Sphere	18880	9442	12	194.5△ /s	1.53 s
Brain	38034	19018	15	181.1△ /s	1.87 s
Heart	25412	12708	14	175.3△ /s	1.78 s
Cylinder	110789	221574	25	162.4△ /s	3.11 s
Airliner	113985	227521	22	173.5△ /s	2.62 s

experiments. As the Flow Charts framework works in the texture space, the runtime performance is mostly determined by the complexity of the dense texture algorithm and the size of the texture, while the viewport size is of little influence. The experiments were conducted on a standard Windows PC equipped with 3 Gbytes of RAM, Intel Core 2 Quad 2.66-GHz processor, and dual nVIDIA 8800 GTX in SLI configuration.

The experimental results discussed above demonstrate that the proposed Flow Charts framework achieves high flexibility and yields satisfactory results due to the use of the surface parameterization. This accomplishment, however, comes at the price of a nontrivial implementation. Although we are able to deliver interactive frame rates using the latest graphics hardware, the image space methods are advantageous in terms of the ease of implementation and shear performance. Another strong point of the image-based methods is scalability. In complex flow data sets, it is not uncommon that the discrepancy of cell size is up to two or three orders of magnitude. High zooming factors can therefore be necessary during the visualization session to investigate intricate flow structures on small cells. In our current design, all Flow Chart textures are in the same uniform resolution. Its scalability is thus limited by the amount of available video memory, and a significant portion of it is consumed by potentially less significant cells. The image-based approaches, on the other hand, are less restrictive since the texture is only generated for the visible part of the surface. The consequences of restricting flow visualization to the visible part of the surface are popping artifacts during rotation of the object and the inability to visualize flow with dye advection. Flow Charts overcomes these limitations.

7 Conclusion and Future Work

We have presented a flexible framework supporting various visualization techniques for flows defined over curved surfaces using a novel scheme called *Flow Charts*. The essence of this method is the introduction of overlapping patches and a redirection scheme allowing particle tracing across patches, which requires special treatments not provided by traditional surface parameterization and segmentation schemes found in the graphics literature. We have demonstrated its versatility by showing how existing schemes can be easily adapted to visualize flows on surfaces. Although our approach is not quite as fast as image-space methods, which only generate textures on the visible parts of surfaces, visualization of flows on curved surfaces using Flow Charts is free of view-dependent

TABLE 2
Runtime Performance of GPUFLIC and UFAC [in Frames per Second]

dataset	texture res	GPUFLIC			UFAC		
		particle ttl=1	particle ttl=2	particle ttl=4	advection only	advection + LIC	+ histogram eq.
ICE train	1200x860	22.5 (10.7)	14.9 (7.1)	8.1 (4.5)	61.2 (41.3)	25.0 (17.2)	12.8 (7.1)
	1600x1140	19.7 (9.4)	10.2 (6.2)	5.7 (2.5)	46.5 (23.6)	27.5 (16.3)	11.3 (6.6)
Delta wing	1200x960	21.4 (14.7)	12.8 (6.1)	7.2 (3.8)	64.7 (45.0)	27.2 (18.4)	11.3 (6.2)
	1600x1280	10.1 (7.2)	5.8 (3.1)	4.4 (2.5)	51.2 (26.3)	27.1 (17.6)	15.5 (8.8)
Sphere	1200x1500	28.8 (15.2)	16.4 (7.2)	9.1 (5.5)	57.3 (25.4)	34.5 (16.2)	23.1 (12.7)
	1600x1470	11.2 (6.1)	7.5 (4.8)	3.6 (1.5)	41.1 (20.5)	31.8 (14.2)	16.8 (7.9)
Brain	1200x1100	25.3 (16.7)	17.4 (8.8)	10.6 (5.1)	52.3 (29.1)	21.4 (17.7)	16.1 (8.2)
	1600x1470	15.3 (7.0)	9.2 (4.9)	5.2 (3.5)	52.1 (29.5)	24.6 (14.9)	18.5 (19.4)
Heart	1200x1040	28.8 (15.6)	17.7 (10.5)	11.8 (5.9)	61.3 (35.5)	25.4 (12.9)	19.2 (9.0)
	1600x1392	18.5 (8.8)	10.2 (6.1)	7.4 (4.0)	54.4 (28.4)	18.5 (8.1)	11.2 (5.2)
Cylinder	1200x1040	29.5 (16.2)	15.4 (8.1)	10.7 (6.6)	57.4 (28.2)	18.3 (9.9)	11.5 (6.8)
	1600x1392	18.4 (8.9)	10.2 (5.2)	6.3 (4.4)	46.1 (25.5)	15.2 (7.5)	8.5 (5.1)
Airliner	1200x1040	31.0 (14.4)	18.6 (9.2)	11.4 (5.2)	52.1 (28.0)	14.3 (6.9)	7.4 (4.8)
	1600x1392	18.5 (9.4)	10.1 (6.3)	6.3 (3.5)	54.3 (25.0)	17.8 (9.2)	8.9 (5.3)

The numbers in parentheses are with dye advection enabled (one 10 × 10 injection site).

artifacts and is still achieving interactive frame rates. It can also correctly depict flow structures across occluded parts of the surface due to the global representation of flow made possible by surface parameterization, which is particularly true in the case of dye advection.

In the future, we want to further experiment with other techniques using this framework and incorporate more sophisticated mesh segmentation/parameterization schemes to further improve the performance and visual quality. Specifically, we would like to investigate the use of nonuniform chart sampling rates to achieve better scalability. More sophisticated chart packing schemes can also be used to increase the utilization of texture space, which is a relatively scarce resource on the graphics hardware. Last, we believe that the Flow Charts representation of flows on curved surfaces are not limited to texture-based flow visualization. Other methods, such as topology analysis and feature detection on curved surfaces, are applications that could benefit from using Flow Charts.

ACKNOWLEDGMENTS

The authors would like to thank Miriah Meyer for providing adaptive surface sampling software. This work was made possible by the following research Grants: US National Science Foundation CNS-0551724, CCF-0541113, IIS-0513212; Department of Energy ASC Alliance C-SAFE, SciDAC-VACET; and NIH/NCRR Center for Integrative Biomedical Computing, P41-RR12553-08.

REFERENCES

[1] Cgal, *Computational Geometry Algorithms Library,* http://www.cgal.org, 2008.
[2] Gts, *GNU Triangulated Surface Library,* http://gts.sourceforge.net, 2006.
[3] H. Battke, D. Stalling, and H.-C. Hege, "Fast Line Integral Convolution for Arbitrary Surfaces in 3D," *Visualization and Math.: Experiments, Simulations and Environments.* Springer-Verlag New York, pp. 181-192, 1997.
[4] B. Cabral and L.C. Leedom, "Imaging Vector Fields Using Line Integral Convolution," *Proc. ACM SIGGRAPH '93,* pp. 263-270, 1993.
[5] N.A. Carr, J. Hoberock, K. Crane, and J.C. Hart, "Rectangular Multi-Chart Geometry Images," *Proc. Eurographics Symp. Geometry Processing (SGP '06),* pp. 181-190, 2006.
[6] D. Eppstein, "Updating Widths and Maximum Spanning Trees Using the Rotating Caliper Graph," Technical Report ICS-TR-93-18, Univ. of California, 1993.
[7] M.S. Floater and K. Hormann, "Parameterization of Triangulations and Unorganized Points," *Tutorials on Multiresolution in Geometric Modelling, Math. and Visualization,* A. Iske, E. Quak, and M.S. Floater, eds., Springer, pp. 287-316, 2002.
[8] M.S. Floater and K. Hormann, "Surface Parameterization: A Tutorial and Survey," *Advances in Multiresolution for Geometric Modelling, Math. and Visualization,* N.A. Dodgson, M.S. Floater, and M.A. Sabin, eds., Springer, pp. 157-186, 2005.
[9] L. Forssell and S. Cohen, "Using Line Integral Convolution for Flow Visualization: Curvilinear Grids, Variable-Speed Animation, and Unsteady Flows," *IEEE Trans. Visualization and Computer Graphics,* vol. 1, no. 2, pp. 133-141, 1995.
[10] S. Gottschalk, M.C. Lin, and D. Manocha, "Obbtree: A Hierarchical Structure for Rapid Interference Detection," *Proc. ACM SIGGRAPH '96,* pp. 171-180, 1996.
[11] X. Gu, S.J. Gortler, and H. Hoppe, "Geometry Images," *ACM Trans. Graphics (TOG),* pp. 355-361, 2002.
[12] T. Igarashi and D. Cosgrove, "Adaptive Unwrapping for Interactive Texture Painting," *Proc. Symp. Interactive 3D Graphics (SI3D '01),* pp. 209-216, 2001.
[13] B. Jobard, G. Erlebacher, and M.Y. Hussaini, "Hardware-Accelerated Texture Advection for Unsteady Flow Visualization," *Proc. Conf. Visualization (VIS '00),* pp. 155-162, 2000.
[14] B. Jobard, G. Erlebacher, and M.Y. Hussaini, "Lagrangian-Eulerian Advection for Unsteady Flow Visualization," *Proc. Conf. Visualization (VIS '01),* pp. 53-60, 2001.
[15] D. Julius, V. Kraevoy, and A. Sheffer, "D-Charts: Quasi-Developable Mesh Segmentation," *Computer Graphics Forum, Proc. Eurographics (EG '05),* vol. 24, pp. 581-590, 2005.
[16] R. Laramee, H. Hauser, H. Doleisch, B. Vrolijk, F. Post, and D. Weiskopf, "The State of the Art in Visualization: Dense and Texture-Based Techniques," *Computer Graphics Forum,* vol. 23, no. 2, pp. 143-161, 2004.
[17] R. Laramee, B. Jobard, and H. Hauser, "Image Space Based Visualization of Unsteady Flow on Surfaces," *Proc. 14th IEEE Visualization (VIS '03),* pp. 18-25, 2003.
[18] 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, 2004.
[19] S. Lefebvre and H. Hoppe, "Appearance-Space Texture Synthesis," *ACM Trans. Graphics,* vol. 25, no. 3, pp. 541-548, 2006.
[20] B. Levy, S. Petitjean, N. Ray, and J. Maillot, "Least Squares Conformal Maps for Automatic Texture Atlas Generation," *ACM Trans. Graphics (TOG),* pp. 362-371, 2002.

[21] G.-S. Li, X. Tricoche, and C. Hansen, "GPUFLIC: Interactive and Accurate Dense Visualization of Unsteady Flows," *Proc. Eurographics/IEEE-VGTC Symp. Visualizations (EuroVis '06),* pp. 29-34, 2006.
[22] S.P. Lloyd, "Least Squares Quantization in PCM," *IEEE Trans. Information Theory,* vol. 28, no. 2, pp. 129-136, 1982.
[23] J. Maillot, H. Yahia, and A. Verroust, "Interactive Texture Mapping," *Proc. ACM SIGGRAPH '93,* pp. 27-34, 1993.
[24] X. Mao, Y. Hatanaka, H. Higashida, and A. Imamiya, "Image-Guided Streamline Placement on Curvilinear Grid Surfaces," *Proc. Conf. Visualization (VIS '98),* pp. 135-142, 1998.
[25] X. Mao, M. Kikukawa, N. Fujita, and A. Imamiya, "Line Integral Convolution for 3D Surface," *Visualization in Scientific Computing, Proc. Eurographics Workshop,* pp. 57-69, 1997.
[26] M.D. Meyer, P. Georgel, and R.T. Whitaker, "Robust Particle Systems for Curvature Dependent Sampling of Implicit Surfaces," *Proc. Int'l Conf. Shape Modeling and Applications (SMI '05),* pp. 124-133, 2005.
[27] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "Rectangle-Packing-Based Module Placement," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design (ICCAD '95),* pp. 472-479, 1995.
[28] P.V. Sander, J. Snyder, S.J. Gortler, and H. Hoppe, "Texture Mapping Progressive Meshes," *Proc. ACM SIGGRAPH '01,* pp. 409-416, 2001.
[29] P.V. Sander, Z.J. Wood, S.J. Gortler, J. Snyder, and H. Hoppe, "Multi-Chart Geometry Images," *Proc. Eurographics/ACM SIGGRAPH Symp. Geometry Processing (SGP '03),* pp. 146-155, 2003.
[30] H.-W. Shen and D.L. Kao, "A New Line Integral Convolution Algorithm for Visualizing Time-Varying Flow Fields," *IEEE Trans. Visualization and Computer Graphics,* vol. 4, no. 2, pp. 98-108, 1998.
[31] J. Stam, "Flows on Surfaces of Arbitrary Topology," *ACM Trans. Graphics (TOG),* pp. 724-731, 2003.
[32] J.J. van Wijk, "Spot Noise Texture Synthesis for Data Visualization," *Proc. ACM SIGGRAPH '91,* pp. 309-318, 1991.
[33] J.J. van Wijk, "Image Based Flow Visualization," *ACM Trans. Graphics,* pp. 745-754, 2002.
[34] J.J. van Wijk, "Image Based Flow Visualization for Curved Surfaces," *Proc. 14th IEEE Visualization (VIS '03),* pp. 123-130, 2003.
[35] D. Weiskopf, "Dye Advection without the Blur: A Level-Set Approach for Texture-Based Visualization of Unsteady Flow," *Computer Graphics Forum,* vol. 23, no. 3, pp. 479-488, 2004.
[36] D. Weiskopf, "Iterative Twofold Line Integral Convolution for Texture-Based Vector Field Visualization," *Math. Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration,* 2007.
[37] D. Weiskopf, G. Erlebacher, and T. Ertl, "A Texture-Based Framework for Spacetime-Coherent Visualization of Time-Dependent Vector Fields," *Proc. 14th IEEE Visualization Conf. (VIS '03),* pp. 107-114, 2003.
[38] D. Weiskopf and T. Ertl, "A Hybrid Physical/Device-Space Approach for Spatio-Temporally Coherent Interactive Texture Advection on Curved Surfaces," *Proc. Conf. Graphics Interface (GI '04),* pp. 263-270, 2004.
[39] D. Weiskopf, F. Schramm, G. Erlebacher, and T. Ertl, "Particle and Texture-Based Spatiotemporal Visualization of Time-Dependent Vector Fields," *Proc. IEEE Visualization Conf. (VIS '05),* pp. 639-646, 2005.

Guo-Shi Li received the BS degree in computer science from the National Taiwan University in 1999 and the MS degree in computer science from the Ohio State University in 2003. He is currently a PhD student in the School of Computing and a member of the Scientific Computing and Imaging Institute, University of Utah. He received the NVIDIA fellowship award in 2004. His research interests include scientific visualization, interactive rendering, and GPU techniques.

Xavier Tricoche received the Engineer's degree in computer science from École Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble (ENSIMAG), the MS degree in applied mathematics from the Université Joseph Fourier, Grenoble, France, in 1998, and the PhD degree in computer science from the University of Kaiserslautern, Kaiserslautern, Germany, in 2002. He is an assistant professor of computer science at Purdue University. Prior to that, he was with the University of Utah as a postdoctoral fellow in the Scientific Computing and Imaging Institute from 2004 to 2006 and as a research assistant professor in the School of Computing from 2006 to 2007. His research interests include topological methods, structural analysis of vector and tensor fields, flow visualization, medical image analysis, interactive data exploration, and computational steering. He is a member of the IEEE.

Daniel Weiskopf received the MSc (Diplom) degree in physics and the PhD degree in physics from Eberhard-Karls-Universität Tübingen, Tübingen, Germany, and the Habilitation degree in computer science from the Universität Stuttgart, Stuttgart, Germany. From 2005 to 2007, he was an assistant professor of computing science at Simon Fraser University, Burnaby, Canada. Since 2007, he has been a professor of computer science in the Visualization Research Center, Universität Stuttgart (VISUS) and in 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 relativities. He is member of the ACM SIGGRAPH, the Gesellschaft für Informatik, and the IEEE Computer Society.

Charles (Chuck) Hansen is a professor of computer science in the School of Computing and an associate director of the Scientific Computing and Imaging Institute, University of Utah. He joined the Computer Science faculty in 1997. From 1989 to 1997, he was a technical staff member in the Advanced Computing Laboratory (ACL), Los Alamos National Laboratory, where he formed and directed the visualization efforts in the ACL. He was a Bourse de Chateaubriand postdoctoral fellow at the Institut National de Recherche en Informatique et en Automatique (INRIA), Rocquencourt, Le Chesnay, France, in 1987 and 1988. From Fall 2004 to Spring 2005, he was a visiting professor at ARTIS/GRAVIR IMAG/INRIA. He was awarded with the IEEE Technical Committee on Visualization and Computer Graphics Visualization Technical Achievement Award in 2005 in recognition of his seminal work on tools for understanding large-scale scientific data sets. He has been working on the visualization and analysis of large-scale scientific data for the past 17 years. His research has made contributions to the fields of scientific visualization, computer graphics, parallel computation, and computer vision. He has published more than 100 peer-reviewed journal and conference papers and has been a coauthor of three papers recognized with "Best Paper Awards" at the IEEE Visualization Conference (1998, 2001, 2002). He is a senior member of the IEEE.