Eurographics Symposium on Geometry Processing 2012
Eitan Grinspun and Niloy Mitra
(Guest Editors)

Volume 31 (2012), Number 5

Stream Surface Parametrization by Flow-Orthogonal Front Lines

Maik Schulze[1], Tobias Germer[1,2], Christian Rössl[1], and Holger Theisel[1]

[1]University of Magdeburg, Germany
[2]think-cell Software GmbH, Germany

Abstract

The generation of discrete stream surfaces is an important and challenging task in scientific visualization, which can be considered a particular instance of geometric modeling. The quality of numerically integrated stream surfaces depends on a number of parameters that can be controlled locally*, such as time step or distance of adjacent vertices on the front line. In addition there is a parameter that cannot be controlled locally: stream surface meshes tend to show high quality, well-shaped elements only if the current front line is "*globally*" approximately perpendicular to the flow direction. We analyze the impact of this geometric property and present a novel solution – a stream surface integrator that forces the front line to be perpendicular to the flow and that generates quad-dominant meshes with well-shaped and well-aligned elements. It is based on the integration of a scaled version of the flow field, and requires repeated minimization of an error functional along the current front line. We show that this leads to computing the 1-dimensional kernel of a bidiagonal matrix: a linear problem that can be solved efficiently. We compare our method with existing stream surface integrators and apply it to a number of synthetic and real world data sets.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations

1. Introduction

Stream surfaces are a standard approach for the visualization of flow data described by vector fields. Carefully chosen and accurately integrated stream surfaces are known to give valuable insight into the behavior of flow phenomena as well as dynamical systems. The practical generation of stream surfaces is essentially a surface meshing problem, however, it is different to the "usual" meshing setting in geometry processing.

Given an n-dimensional vector field $\mathbf{v}(\mathbf{x})$ and an open parametric seed curve $\mathbf{s}_0(s)$ with $s \in [s_0, s_1]$, a *stream surface* $\mathbf{s}$ can be written as a parametric surface $\mathbf{s}(s,t)$ fulfilling

$$\frac{\partial \mathbf{s}(s,t)}{\partial t} = \mathbf{v}(\mathbf{s}(s,t)) \tag{1}$$

with the initial condition $\mathbf{s}(s,0) = \mathbf{s}_0(s)$ for $s \in [s_0, s_1]$. The isoparametric lines $s = \text{const}$ are *stream lines* and $t = \text{const}$ give *time lines*. In particular, time $t = 0$ gives the seed curve $\mathbf{s}_0(s)$. In general, neither a closed form nor an implicit description of $\mathbf{s}$ exists, so that robust numerical integration schemes starting from the seed curve are necessary. Let $\mathbf{s}_s$ and $\mathbf{s}_t = \mathbf{v}(\mathbf{s})$ be the partial derivative vectors of $\mathbf{s}$. While the numerical integration of stream lines is well-understood, stream surface integration is still challenging because $\mathbf{s}$ tends to be badly parametrized with increasing integration time. In fact, the magnitude of $\mathbf{s}_s$ can become extremely large (or small) during the integration, meaning that tiny parts of the seed curve produce large areas of the stream surface (or vice versa). This lead to the development of algorithms for stream surface integration in the scientific visualization community. Mainly, these are advancing front algorithms that combine different operations for

- robust adaptive stream line integration,
- heuristics for adaptive refinement/coarsening of the current front line, and
- (in some cases) a special treatment near critical points.

This way, these approaches provide adaptive solutions for large magnitudes of $\mathbf{s}_s$ and $\mathbf{s}_t$, respectively, and give impressive results in many cases. Nevertheless, they fail in other (and even rather simple) cases in the sense that the result-

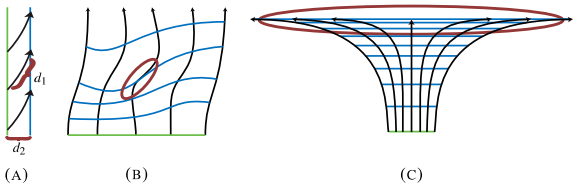

Figure 1: *Small angle problem for stream line integration. Stream lines are black, time lines blue, and seed curves green. Regions of small angle between stream lines and time lines are red.* (A) *Long integration of* $\mathbf{v}$ *(here* d_1*) leads to a small progress of the stream surface (here* d_2*).* (B) *The small angle problem can appear for a seed curve perpendicular to the flow and even in the absence of critical points.* (C) *The small angle problem induced by a saddle point appears not only close to the saddle but in a larger region.*

ing meshes tend to produce too many badly shaped triangles, which limits accuracy, quality, and performance of the approaches. This is due to the fact that all operations mentioned above are *local* , i.e., they use only local information to perform the adaptive steps. However, there are imperfections in the parametrization of $\mathbf{s}$ that can only be treated by a *global* approach to stream surface integration, i.e., the decisions about the next integration steps are based on a global analysis of the current stream surface front line and the vector field. Such problems occur in regions where stream lines and time lines intersect in a small angle, even though the magnitudes of $\mathbf{s}_s$ or $\mathbf{s}_t$ behave well. We call this the *small angle problem* for stream line integration.

The small angle problem creates different challenges for stream surface integration. Firstly, long stream line integrations of points on the front line are necessary to obtain a small progress of the stream surface, leading to an accumulation of numerical error for stream line integration. Figure 1(A) illustrates this. Secondly, the small angle problem leads to badly shaped triangles, such that an additional post-processing, e.g., remeshing, is required. Thirdly, for time-varying vector fields, $\mathbf{v}$ and the front line can become parallel, leading to a non-regular parametrization of $\mathbf{s}$ and degenerate triangles in the mesh. Note that the small angle problem cannot be solved by a careful selection of the seed curve: even if the seed curve is perpendicular to $\mathbf{v}$ everywhere, it can run into the small angle problem during the integration. Figure 1(B) illustrates this. Also note that a simple normalization of $\mathbf{v}$ does not solve the problem either. One prominent cause for the small angle problem is when the front line hits a saddle point. In fact, particular solutions which strive for a robust handling near critical points were proposed.

However, a saddle does not only create the small angle problem in its local neighborhood but also in larger areas. Figure 1(C) illustrates this. Moreover, the small angle problem can occur even in absence of any saddle, as illustrated in Figure 1(B).

In this paper we present a solution to the small angle problem. From the considerations made above it becomes clear that this requires a global, adaptive approach. We propose an advancing front algorithm that forces the front line to be orthogonal to the flow, i.e., the angle between stream line and time line should be close to 90 degrees. This can also be interpreted as a conformal parametrization of stream surfaces, which yields discrete surface meshes consisting of quad-dominant structures with well-shaped, flow-aligned elements. We achieve this by integrating not $\mathbf{v}$ but a scaled version $\alpha\mathbf{v}$, where $\alpha(\mathbf{x})$ is a suitable scalar field. This is essentially a reparametrization of the stream surface w.r.t. time. We define α to enforce a *self-correcting front line*: starting with an arbitrary seed curve, the angle between the current front line and $\mathbf{v}$ is forced towards being orthogonal during the integration for all points of the front (Section 3). We show that for a piecewise linear approximation of the front line (Section 4) α can be computed as the solution to a bidiagonal linear system (Section 5). This results in a combination of the advancing front integration of the surface with an advancing front integration of the scalar field α. We discuss adaptive mesh generation and implementation details (Section 6) and show results together with a comparison to state-of-the-art stream surface meshing for a number of synthetic and real flow fields (Section 7). Finally, we discuss benefits and limitations of our approach (Section 8).

2. Related Work

In flow visualization, Hultquist [Hul92] pioneered the development of stream surface integrators: this work is still the basis for state-of-the-art methods. Hulquist's method consists of a marching front algorithm that builds a triangle mesh by particle integration and recursively adding triangles at the front. Note that this approach already considers the small angle problem, at least indirectly: the decision on which parts of the front line ought to advance next in the recursion implicitly keeps the front somewhat perpendicular to the flow. Note that this decision is based on local criteria. The price for this scheme is that the algorithm is recursive (or stack-based), making a parallel implementation (e.g., on the GPU) impossible, which is a drawback for real-time meshing in interactive application.

Stalling [Sta98] extends this method by a still recursive algorithm and provides a special treatment to areas near saddle points, where stream lines diverge. (A rigorous treatment of critical points and stream surface topology is given in [SRWS10].)

Garth et al. [GTS*04, GKT*08] present several improvements that avoid the recursion but in turn cannot address the small angle problem. This results in front lines that are not perpendicular to the flow and leads to degenerated meshes. An approach to improve the approximation order and subdivision at the current front is proposed in [SWS09]. Scheuermann et al. [SBH*01] present a stream surface integrator

that relies on exact solutions of the integration for piecewise linear fields. Schafhitzel et al. [SST*07] present a GPU-accelerated method based on the integration of particles. Peikert and Sadlo [PS09] put focus on topologically relevant structures, their method extracts complete and crack-free stream surfaces.

Most of the approaches mentioned above do not consider the small angle problem. Exceptions are in [Hul92] and the extension [Sta98], which produce approximately orthogonal front lines at the cost of using recursive algorithms. Another notable exception presented by McLoughlin et al. [MLZ09] measures local shear angles to locally adapt the progress of the front line for generation of quad structures. While the basic idea of adapting the speed of the front line is also found in our method, the means to achieve this are very different. This method modifies the step size of the integrator which impacts the numerical integration scheme and was implemented for linear Euler steps and Runge-Kutta integrators of order 2. All of the aforementioned methods rely on purely local observations to tackle the small angle problem. This, however, is generally not sufficient to solve it and in contrast to our approach, which examines a global setting.

The remaining approaches either treat the impact of the small angle problem only in the vicinity of saddle points [SRWS10], or they completely disregard it.

The problem of stream surface integration has also been regarded in the context of dynamical systems. In fact, algorithms that extract stable and unstable manifolds of dynamical systems are actually stream surface integrators. Doedel et al. [DKK91] present an approach based on a continuation of trajectories. Krauskopf et al. [KO99, KO03] present approaches based on an approximation of geodesic level sets. Henderson [Hen05] computes so-called fat trajectories. A PDE formulation was given in [GV04], and Dellnitz et al. [DH97] present a box covering approach. We refer to the survey [KOD*05] for an overview of existing methods for computing (un)stable manifolds of vector fields. The above mentioned methods generally produce high-quality surfaces, however, they are suited for off-line processing only and are far from being interactive. In contrast to our approach, none of these was ever designed for real-time demands, which are common in flow visualization and exploration. Our examples include the well-known Lorenz-attractor (see Section 7), which shows that our approach can compete with such off-line methods in terms of mesh quality.

Finally, the approach of enforcing orthogonality has recently been used in a different context: Schulze et al. [SRGT12] propose as-perpendicular-as-possible surfaces for flow visualization: in contrast to stream surfaces, which are tangential to the flow, these are surfaces that are approximately perpendicular to a steady vector field.

3. Flow-Orthogonal Frontlines

Our approach consists in constructing a reparametrization $\bar{\mathbf{s}}$ of the stream surface $\mathbf{s}$. This is achieved by modifying the integration: instead of integrating $\mathbf{v}$ using (1), we integrate a scaled vector field $\alpha\mathbf{v}$ where $\alpha(\mathbf{x},t)$ is a suitable scalar field. The field α is evaluated only *on* the stream surface, therefore we write $\alpha(s,t) := \alpha(\mathbf{s}(s,t),t)$. We are interested in the stream surface $\bar{\mathbf{s}}(s,t)$ that satisfies

$$\frac{\partial\,\bar{\mathbf{s}}(s,t)}{\partial t} = \alpha(s,t)\,\mathbf{v}(\bar{\mathbf{s}}(s,t))$$

with the initial condition $\bar{\mathbf{s}}(s,0) = \mathbf{s}_0(s)$. Since $\bar{\mathbf{s}}$ is a reparametrization of $\mathbf{s}$ w.r.t. the time parameter t, the image of $\bar{\mathbf{s}}$ and $\mathbf{s}$ have the same shape. (Here, and in the following we assume regular surfaces, i.e., non-vanishing partials.) In order to locally capture the angle between stream lines and time lines, we consider the (signed) error function

$$e(s,t) \;=\; \bar{\mathbf{s}}_s^T\,\mathbf{v}\,. \tag{2}$$

We assume that time lines of $\alpha\,\mathbf{v}$ are arc-length parametrized, i.e., $\|\bar{\mathbf{s}}_s\| = 1$. Then the error e is a measure of how perpendicular stream lines and time lines are: perfectly perpendicular lines result in a vanishing error $e = 0$. The dependence of the error on $\|\mathbf{v}\|$ can be interpreted as follows: the larger the magnitude $\|\mathbf{v}\|$, the more e penalizes non-perpendicularity of $\mathbf{s}_s$ and $\mathbf{v}$. This corresponds to the fact that for large $\|\mathbf{v}\|$ the small angle problem has a larger impact on the stream surface integration.

In general, the error function e is *nonzero*. We want to define the scalar field α such that e is corrected towards zero during the integration of $\alpha\,\mathbf{v}$. We express this by demanding

$$\frac{\partial e}{\partial t} + e = 0\,. \tag{3}$$

This can be interpreted as follows: the larger the error e at time t the more it will be damped towards zero by the differential $\frac{\partial e}{\partial t}$ at the next time step $t + \partial t$. This way, e converges towards 0 during the integration. What remains to be done is to find conditions on α such that (3) holds. Using

$$\frac{\partial e}{\partial t} \;=\; \left(\frac{\partial\,\bar{\mathbf{s}}_s}{\partial t}\right)^T\mathbf{v} \;+\; \bar{\mathbf{s}}_s^T\,\frac{\partial \mathbf{v}}{\partial t} \tag{4}$$

and keeping in mind that the directional derivative of $\mathbf{v}$ in direction $\mathbf{r}$ can be written in terms of the Jacobian matrix $\mathbf{J}$ as $\mathbf{J}\,\mathbf{r}$, we obtain for $\mathbf{r} = \alpha\mathbf{v}$

$$\frac{\partial \mathbf{v}}{\partial t} \;=\; \mathbf{J}(\alpha\,\mathbf{v})$$

and hence

$$\begin{aligned}\frac{\partial\,\bar{\mathbf{s}}_s}{\partial t} \;&=\; \frac{\partial\,\bar{\mathbf{s}}_t}{\partial s} \;=\; \frac{\partial(\alpha\mathbf{v})}{\partial s} \;=\; \frac{\partial\alpha}{\partial s}\,\mathbf{v} + \alpha\,\frac{\partial \mathbf{v}}{\partial s}\\ &=\; \alpha_s\,\mathbf{v} + \alpha\,\mathbf{J}\,\bar{\mathbf{s}}_s\,.\end{aligned} \tag{5}$$

Inserting (5) into (4) and this into (3) gives

$$\alpha\,\left(\bar{\mathbf{s}}_s^T(\mathbf{J}+\mathbf{J}^T)\mathbf{v}\right) \;+\; \alpha_s\,\left(\mathbf{v}^T\mathbf{v}\right) \;+\; \bar{\mathbf{s}}_s^T\mathbf{v} \;=\; 0\,.$$

This is the condition on α that ensures flow-orthogonality for the integration of $\alpha\,\mathbf{v}$. As the partial α_t is not involved in

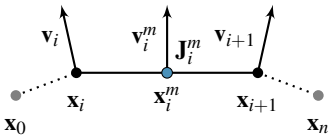

Figure 2: *Quantities required to estimate the local discrete error for a segment $[\mathbf{x}_i, \mathbf{x}_{i+1}]$ of the current front line.*

this condition, the computation of α can be combined with an advancing front integration of $\mathbf{s}$: regard the current front line at $t = \hat{t}$. Then the computation of α along $\mathbf{s}(s, \hat{t})$ is simply an initial value problem because $\mathbf{v}$ and $\mathbf{J}$ are known along the front. This implies that there remains one degree of freedom for the solution: $\alpha(s_0, \hat{t}) \in \mathbb{R}$ can be fixed as boundary condition for a certain parameter s_0.

4. Discretization

In the discrete setting the current front line at time $\hat{t}$ is approximated by a polyline with vertices $(\mathbf{x}_1, ..., \mathbf{x}_n)$, $\mathbf{x}_i \in \mathbb{R}^3$, where we assume that the vertex positions are distributed approximately uniformly along the front. (Section 6 details how such discrete arc-length parametrization is maintained for the resulting surface mesh.) The new front line at the next time step is obtained by integration along the scaled vector field $\alpha(\mathbf{x}_i, \hat{t})\,\mathbf{v}(\mathbf{x}_i)$. We search for a piecewise linear scalar field α, such that this new front line is orthogonal to the flow. The field α is represented by a vector $\alpha = (\alpha_1, ..., \alpha_n) \in \mathbb{R}^n$, where each value α_i is associated with the vertex $\mathbf{x}_i$.

We define the piecewise constant error function e at each segment $[\mathbf{x}_i, \mathbf{x}_{i+1}], i = 1, \ldots n-1$, of the line by evaluating the error in the middle $\frac{1}{2}(\mathbf{x}_i + \mathbf{x}_{i+1})$ of each segment. We set

$$\mathbf{v}_i = \mathbf{v}(\mathbf{x}_i)\ ,\ \mathbf{x}_i^m = \frac{\mathbf{x}_{i+1} + \mathbf{x}_i}{2}\ ,\ \mathbf{v}_i^m = \mathbf{v}(\mathbf{x}_i^m)\ ,\ \mathbf{J}_i^m = \mathbf{J}(\mathbf{x}_i^m)\ .$$

Then the discrete counterpart of (2) defines the error at $\mathbf{x}_i^m$ as

$$e_i = (\mathbf{x}_{i+1} - \mathbf{x}_i)^T\, \mathbf{v}_i^m\ . \tag{6}$$

Figure 2 illustrates the setup. We demand

$$\frac{\partial e_i}{\partial t} + e_i = 0\ . \tag{7}$$

Using

$$\begin{aligned}
\frac{\partial e_i}{\partial t} &= \frac{\partial (\mathbf{x}_{i+1} - \mathbf{x}_i)}{\partial t}^T \mathbf{v}_i^m + (\mathbf{x}_{i+1} - \mathbf{x}_i)^T \frac{\partial\, \mathbf{v}_i^m}{\partial t} \\
\frac{\partial\, \mathbf{v}_i^m}{\partial t} &= \mathbf{J}_i^m\, \frac{\alpha_i \mathbf{v}_i + \alpha_{i+1} \mathbf{v}_{i+1}}{2} \\
\frac{\partial (\mathbf{x}_{i+1} - \mathbf{x}_i)}{\partial t} &= \alpha_{i+1}\, \mathbf{v}_{i+1} - \alpha_i\, \mathbf{v}_i\ ,
\end{aligned}$$

(the last equality comes from a local linearization of $\mathbf{v}$) equation (7) can be written as

$$\alpha_i\, p_i + \alpha_{i+1}\, q_i + r_i = 0$$

with

$$\begin{aligned}
p_i &= \tfrac{1}{2}\, (\mathbf{x}_{i+1} - \mathbf{x}_i)^T\ \mathbf{J}_i^m\, \mathbf{v}_i - \mathbf{v}_i^T \mathbf{v}_i^m \\
q_i &= \tfrac{1}{2}\, (\mathbf{x}_{i+1} - \mathbf{x}_i)^T\ \mathbf{J}_i^m\, \mathbf{v}_{i+1} + \mathbf{v}_{i+1}^T\, \mathbf{v}_i^m \\
r_i &= (\mathbf{x}_{i+1} - \mathbf{x}_i)^T\ \mathbf{v}_i^m
\end{aligned}$$

for $i = 1, ..., n-1$.

Hence, finding α results in finding a solution to the underdetermined linear system $\mathbf{M}\alpha = \mathbf{r}$ with $\mathbf{M} \in \mathbb{R}^{(n-1)\times n}$, which has the structure

$$\begin{pmatrix} p_1 & q_1 & & & \\ & p_2 & q_2 & & \\ & & \ddots & \ddots & \\ & & & p_{n-1} & q_{n-1} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{pmatrix} = \begin{pmatrix} -r_1 \\ -r_2 \\ \vdots \\ -r_{n-1} \end{pmatrix}\ .$$

5. Finding the Scalar Field α

The linear system $\mathbf{M}\alpha = \mathbf{r}$ (4) is underdetermined, and solutions $\alpha \in \mathbb{R}^n$ are found in the affine subspace $\alpha^P + \mu\mathbf{k}$, where α^P is any particular solution, and $\mathbf{k}$ is a vector spanning the 1-dimensional kernel $\{\alpha \,|\, \mathbf{M}\alpha = \mathbf{0}\}$. Given $\mathbf{M}$ and $\mathbf{r}$, there are different ways to determine $\mathbf{k}$ and α^P. The numerically most stable way is using the factorization $\mathbf{M}^T = \mathbf{QR}$ with an orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{n\times n}$ and upper triangular matrix $\mathbf{R}^{n\times(n-1)}$: then we obtain $\mathbf{k}$ as the unit vector $(0, \ldots, 0, 1)\,\mathbf{Q}$, i.e., the n-th column of $\mathbf{Q}$, and we chose α^P as the least-norm solution $\alpha^L = \mathbf{Q}_{[n-1]}\mathbf{R}_{[n-1]}^{-T}\mathbf{r}$, i.e, $||\alpha^L||$ is minimal for all solutions to $\mathbf{M}\alpha = \mathbf{r}$. The index $[\cdot]$ denotes the restriction to the first $n-1$ columns of $\mathbf{Q}$ or rows of $\mathbf{R}$.

Due to the bidiagonal structure of $\mathbf{M}$ the necessary computations can be carried out efficiently. We apply $n-1$ Givens plane rotations to $\mathbf{M}^T$ to construct $\mathbf{R}$. Note that the upper triangular $\mathbf{R}$ is also bidiagonal, i.e., we transform subdiagonal and diagonal of $\mathbf{M}^T$ to diagonal and superdiagonal of $\mathbf{R}$. Then we solve the bidiagonal linear system $\mathbf{y} = \mathbf{R}_{[n-1]}^T\mathbf{r}$. And we finally obtain $\alpha^L = \mathbf{Q}_{[n-1]}\,\mathbf{y}$ and $\mathbf{k} = (0, \ldots, 0, 1)\,\mathbf{Q}$, where the orthogonal transformation is carried out as $n-1$ plane rotations, i.e., we do not generate the matrix $\mathbf{Q}$ explicitly. This way we require a total of $26n$ flops to compute both, the unit vector $\mathbf{k}$ spanning the kernel of $\mathbf{M}$ and the least-norm solution α^L. We can give an interpretation for different solutions $\alpha = \alpha^L + \mu\,\mathbf{k}$. We start with the computed least-norm solution α^L. This is the solution closest to the origin, which means that choosing $\alpha = \alpha^L$ will minimize the discrete error $\sum_i e_i$ defined in (6) most rapidly. This can be seen from two perspectives: for a given number of integration steps we obtain the smallest error. And likewise for a given error bound we require the least number of integration steps, i.e., using the least-norm solution $\alpha = \alpha^L$ yields fastest convergence towards a flow-orthogonal front line.

However, minimizing the error rapidly does not necessarily advance the front line significantly enough, a condition which is required to evolve the stream surface. In fact, these two goals are competing. The simplest example is $\alpha^L = \mathbf{0}$, hence $e_i = 0$, and the front line is orthogonal to the flow. Even more, choosing any $\alpha \neq \mathbf{0}$ may increase the error – but at the same time the surface evolution stopped for $\alpha = \mathbf{0}$.

If we want to advance the front line we have to ensure that $\alpha_i > 0$ for all or for entries $i = 1, \ldots, n$. There is no

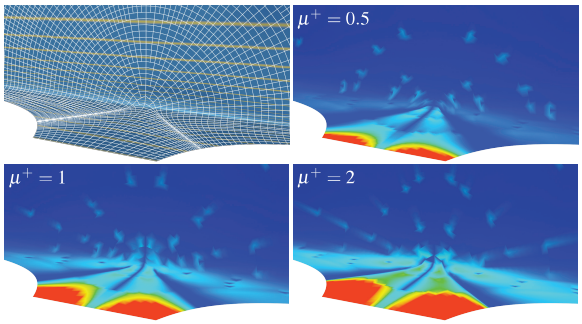

Figure 3: *Top left to bottom right: Flow-orthogonal front lines near repelling saddle node. The error e is color-coded from blue=low to red=high for different* μ^+.

guarantee that such α exists. We apply a heuristic and relax the condition from having all to having as many positive entries as possible and proceed as follows: count the number p of positive entries in $\mathbf{k}$. Set $\mathbf{k}^+ = \mathbf{k}$ if $p > n/2$ and $\mathbf{k}^+ = -\mathbf{k}$ otherwise, i.e., $\mathbf{k}^+$ spans the kernel of $\mathbf{M}$ and is oriented in direction with more positive coordinates. Then we have $\alpha = \alpha^L + \mu^+\mathbf{k}^+$, and obtain the scalar design parameter $\mu^+ \geq 0$: for $\mu^+ = 0$ the front line will align orthogonally to the flow at the cost of small or no advection. For increasing μ we can obtain more and more positive entries and larger absolute magnitudes of α_i. Hence the front line will be advected quicker with the flow at the cost of being less quickly aligned orthogonally to the flow. In practice, we suggest a standard value of $\mu^+ = 1$, which works fine for our experiments. If required, the parameter μ^+ can be changed interactively.

The effect of changing μ^+ on the minimization of the error can be seen in Figure 3 which shows stream surfaces near a 3D repelling node saddle. The surface mesh is locally dense, where α_s is high. The color-coded visualizations show the error e defined in Equation (2) for different values of μ^+: lower values of μ^+ lead to faster minimization of e at the cost of a slower front line advancement and vice versa for higher values.

Note that this heuristic does not guarantee absence of foldovers in the final surface mesh. In our experiments we never observed any foldovers. In case a guarantee is required, one solution is using an orientation-free integrator, as proposed by Theisel et al. [TSW*05] for feature flow fields and eigenvector fields.

6. Mesh Generation

The goal of this work is to generate stream surface meshes with a quad-dominant structure and well-shaped, flow-aligned elements. In practice, triangle meshes are often preferred for direct rendering. Thus, in the following we consider triangulations (or essentially triangulated quads). In order to generate high-quality meshes, an adaptive refinement and handling of vector field singularities is combined with the flow-orthogonal front lines parametrization.

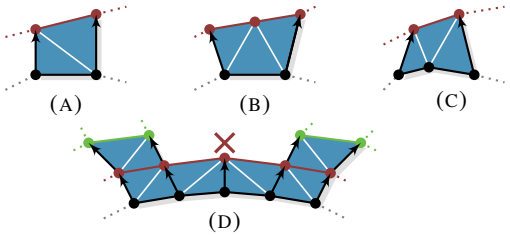

Figure 4: *Mesh generation.* (A) *Normal front advancement.* (B) *Refinement by split.* (C) *Coarsening by merge.* (D) *Surface ripping near saddle. Colors denote different fronts.*

6.1. Front Advancement

The surface is built layer by layer by integrating the front line in the flow $\alpha\mathbf{v}$. The front at time t is also a time line of $\alpha\mathbf{v}$. It is represented by vertices $\mathbf{x}_1(t),\ldots,\mathbf{x}_n(t)$. (We omit parameter t or an additional index, whenever the meaning is clear from the context.) Each new layer is determined by first computing the scalar field α along the front and then applying a fourth-order Runge-Kutta integration for each vertex $\mathbf{x}_i$. We assume that the segments of the front line have an approximately equal length $\ell \approx ||\mathbf{x}_{i+1}(t) - \mathbf{x}_i(t)||$ for $i = 1,\ldots,n-1$. Our goal is to choose the step size h such that the integration step approximates this length as well: $||\mathbf{x}_i(t) - \mathbf{x}_i(t+h)|| \approx \ell$ for $i = 1,\ldots,n$. This means that segment lengths on the time line $\mathbf{x}_i$ should match the segment lengths on the stream lines. The resulting new layer is a quad structure, which is triangulated by inserting diagonals (see Figure 4(A)).

The α-scaling of $\mathbf{v}$ corrects the speed of vertices $\mathbf{x}_i$ such that the new front line is orthogonal to the flow. This is based on the derivations presented in the previous sections that determine the *relative* speed of vertices. For mesh generation we are left with a "global scale" that is represented as the length of the time interval h for integration. (Note that h is not same as and independent of the adaptive step size for the Runge-Kutta integrator.) This scale parameter is determined independently for every integration step of the front line and stays constant during this step. The choice of h determines the (average) segment length $||\mathbf{x}_i(t) - \mathbf{x}_i(t+h)||$ on stream lines. Let ℓ be the prescribed average segment length on the front line. Then we choose h as the ratio

$$h = \min\left\{ \frac{\ell}{|\alpha_i| \cdot ||\mathbf{v}(\mathbf{x}_i(t))||}, i = 1,\ldots,n \right\} .$$

In practice we additionally impose an upper bound to avoid large time intervals. In our implementation, we use $\min\{h, 1\}$. The basic front advancement generates a regular mesh, where edge lengths in s and t direction are bounded and do not vary quickly in s and time t-direction. Note that the actual edge length $||\mathbf{x}_{i+1}(t) - \mathbf{x}_i(t)||$ serves as a weighting term for the error e in the discrete case: longer edges contribute more than shorter edges.

6.2. Front Adaptation

The basic meshing algorithm described above assumes an approximate arc-length parametrization of the current front.

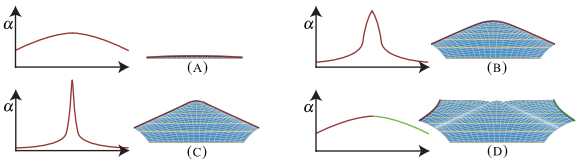

Figure 5: *Four time steps* (A)-(D) *of a stream surface near a saddle in combination with the* α *field. A spike in the scalar field evolves, most distinctive at* (C)*, which guides ripping.*

It is designed to limit lengths of segments on stream lines. However, this does not necessarily mean that the new front line is still arc-length parametrized. We fix this with an additional step that adaptively refines or coarsens the new front line. This way, the mesh is also globally adaptively refined.

The basic idea is simple and includes two local operations: split and merge. If a segment length $||\mathbf{x}_i, \mathbf{x}_{i+1}||$ grows beyond a threshold $\ell_{\max}$, then the segment is *split* into two segments by inserting a new vertex at the barycenter $\frac{1}{2}(\mathbf{x}_i + \mathbf{x}_{i+1})$. Conversely, if the length of two adjacent line segments $||\mathbf{x}_{i-1}, \mathbf{x}_i|| + ||\mathbf{x}_i, \mathbf{x}_{i+1}||$ becomes smaller than threshold $\ell_{\min}$, the algorithm removes the middle vertex merging them into a single segment (see Figure 4(B-C)). We observe that $\ell_{\max} = \frac{3}{2}\ell$ and $\ell_{\min} = \frac{5}{4}\ell$ work well for all our examples. Figure 4 illustrates this process and also shows the subsequent local adaptation of the triangulation.

Note that there exist other sophisticated criteria for splitting and merging (see, e.g., [GKT*08]). Such alternatives may be useful and advantageous depending on the application of the resulting stream surfaces. Their integration into our method is straightforward.

6.3. Ripping

Many stream surface integration methods require special treatment in the vicinity of saddle points [Hul92, Sta98, SBH*01, SRWS10]. In this case, the stream surface contains a separating stream line $\mathscr{S}$ such that stream lines on different sides of $\mathscr{S}$ diverge. In our mathematical framework this corresponds to a peak in $\alpha(s)$, which manifests near the seed point of $\mathscr{S}$ on the front line $\bar{\mathbf{s}}(s,\hat{t})$, and which becomes higher and sharper as the front approaches the saddle until it would degenerate to a Dirac delta impulse. The situation can be described informally as follows: the scalar field α strives to compensate local deviation from flow-orthogonality. When the front line approaches the saddle, the only way to achieve this is to stop integration everywhere, i.e., $\alpha \to 0$ but at the saddle. There we observe $\alpha \to \infty$ as $||\mathbf{v}|| \to 0$. For such configurations, the whole front line does not move anymore during integration.

Figure 5 shows this case. Four different time steps of surface integration are shown. On the left side the scalar field α is shown for the current front line (red and green in the images on the right). The red front line seeded in Figure 5(A) shows a smooth α field because it is almost orthogonal to the vector field. After a few integration steps the α field becomes sharper in the middle as seen in Figure 5(B). Figure 5(c) finally shows the scalar field near the saddle. The peak is detected, and the front line is split into two independent ones, shown in red and green. After that the scalar field α is computed for each front line and evolves smoothly after a few time steps (see Figure 5(D)).

In order to allow for further progress of the integration, we rip the stream surface: we detect the peak in α by testing if the second derivative $\frac{\partial^2 \alpha}{\partial s^2}$ exceeds a given threshold. In the discrete setting, we use finite differences of α_i, and whenever we detect a peak at α_k, we remove the associated vertex $\mathbf{x}_k$ from the front line. Effectively, this splits the current front into two parts, which are further integrated separately as independent front lines. The stream surface appears as ripped near the saddle (Figure 5(D)).

This way, we use the scalar field α along the current front as a *detector for saddles*. This criterion is global to the whole front line in contrast to purely local approaches, like the criterion used by [Hul92]. Non-locality makes our approach more robust than local methods. Peaks are detected reliably without data-dependent tuning of the threshold parameter.

6.4. Implementation

We implemented most stages of our algorithm as parallel primitives executed on the GPU using NVIDIA's CUDA framework. Both, surface and vector field data, reside solely on the GPU. The first step of the iterative front line integration is the computation of the scalar field α. The arising linear system is set up on the GPU and solved on the CPU. This is in fact the only part of our implementation that is not executed in parallel and showed better performance on the CPU. Note that the amount of data to be transferred is small, and neither QR-decomposition nor solving a bidiagonal system benefit from parallel execution (see Section 5). After uploading the computed scalar field α to the GPU, we apply the rip criterion. Note also that the scalar field is not valid anymore for front lines that have been ripped and needs to be recomputed. Afterwards, front line adaptation and integration are performed. Finally, the triangulation between the current and the former front line is build. In summary, the integration and all meshing stages are executed on the GPU.

7. Results

In this section we show stream surface meshes that were generated with our method from synthetic input data and from flows obtained from practical simulations. In addition, we implemented the state-of-the-art methods by Stalling [Sta98] and Garth et al. [GKT*08] (which emerged from [Hul92]) and compare to their results.

Visualization. Red arrows indicate the main flow direction. For stream surfaces, we superimpose the triangle or quad meshes (white wire-frame). Yellow lines show additional

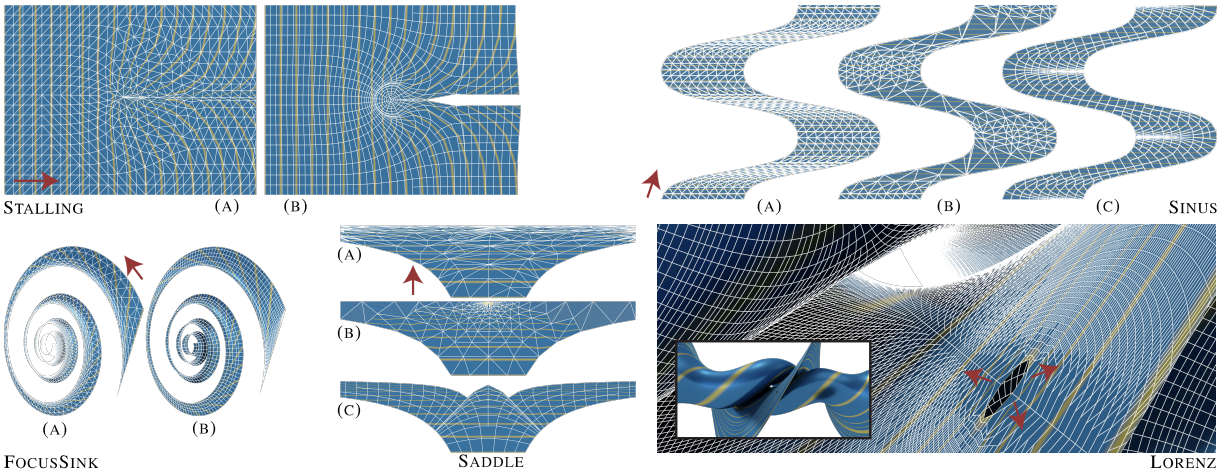

Figure 6: *Results for synthetic data sets.* SADDLE *and* SINUS*: [GKT*08] (A), [Sta98] (B), and ours (C).* STALLING *and* FOCUSSINK*: [Sta98] (C), and ours (B),* LORENZ *attractor: ours.*

time-lines ($t = \text{const}$) of $\mathbf{v}$ seeded at equidistant times. The accompanying video shows our interactive application and further examples.

7.1. Synthetic Data Sets

Figure 6 compares stream surfaces generated by different algorithms – [Sta98], [GKT*08], and our method.

The SADDLE field consists of a planar flow towards a saddle point, and we seed seed stream surfaces from the bottom line. Here, the small angle problem leads to almost parallel stream lines and time lines for a naïve stream surface integration.

The time line based method by Garth et al. [GKT*08] shown as SADDLE (A) produces increasingly dense stream lines with skinny and nearly degenerate triangles. Also, it does not detect the saddle as a topological feature. Stalling's algorithm [Sta98] in SADDLE (B) detects the saddle and creates a consistent mesh in the ε-area around the critical point. As triangles become smaller and smaller due to decreasing velocity, the resulting surface mesh tends to over-tessellation. The result from our method is shown in figure 6 SADDLE (C). Almost all triangles are well-shaped and approximately equally sized. The flow-orthogonal front lines are clearly visible in the wire-frame visualization. The surface mesh is ripped near the saddle, i.e., the front line is split. In this case this results in a $\wedge$-shaped mesh boundary near the saddle. The detection of the critical point and surface ripping work robustly and reliably also for other critical point configurations in the following data sets.

The SINUS data set is modeled as a sinusoidal flow defined by $\mathbf{v}(x,y,z) = (5\,sin(y)^7 + y, 1, 0)$. This field is a good benchmark because constant changes in the angle between streamline and timeline result in a constantly shifting scalar field α. SINUS (A) shows the result from [GKT*08], which tends to produce badly shaped triangles. SINUS (B) shows the result from [Sta98] and illustrates the velocity dependence: high shear areas lead to large triangles and possibly undersampling. Figure 6 SINUS (C) shows the result of our algorithm. The front lines are orthogonal to the flow, and there are only small regions of oversampling.

The STALLING data set is modeled after a planar flow benchmark described in [Sta98] with a main flow direction from left to right. The vector field contains a sink in the center and a saddle on the right boundary of the domain. We compare Stalling's algorithm [Sta98] (A) to our method (B). Note that the vector field had been normalized for Stalling's algorithm to avoid the generation of a large number of small triangles near the critical points. Both algorithms detect the critical points. Stalling's tracer placement closes the gap that emerges from ripping the stream surface. Our basic method lacks such sort of topology-aware meshing. If required, however, an extension is straightforward.

The FOCUSSINK example shows stream surfaces near a focus sink. The seed line on the far right is chosen badly in the sense that it is far from orthogonal to the flow. Here, Stalling's algorithm (A) generates increasingly many triangles of decreasing size because their edge length is coupled to the velocity of the vector field. In contrast, our method in (B) generates a surface mesh with relatively few quads and with low variation in their areas for a prescribed target edge length. For this example, both algorithms adjust the alignment of front lines orthogonal to the flow well.

The LORENZ attractor data set is a prominent benchmark for high-quality, off-line algorithms as used, e.g., in the dynamical systems context. The stream surface generated by our algorithm grows from an open, circular seeding structure. For this complex example we generate high-quality meshes in real-time.

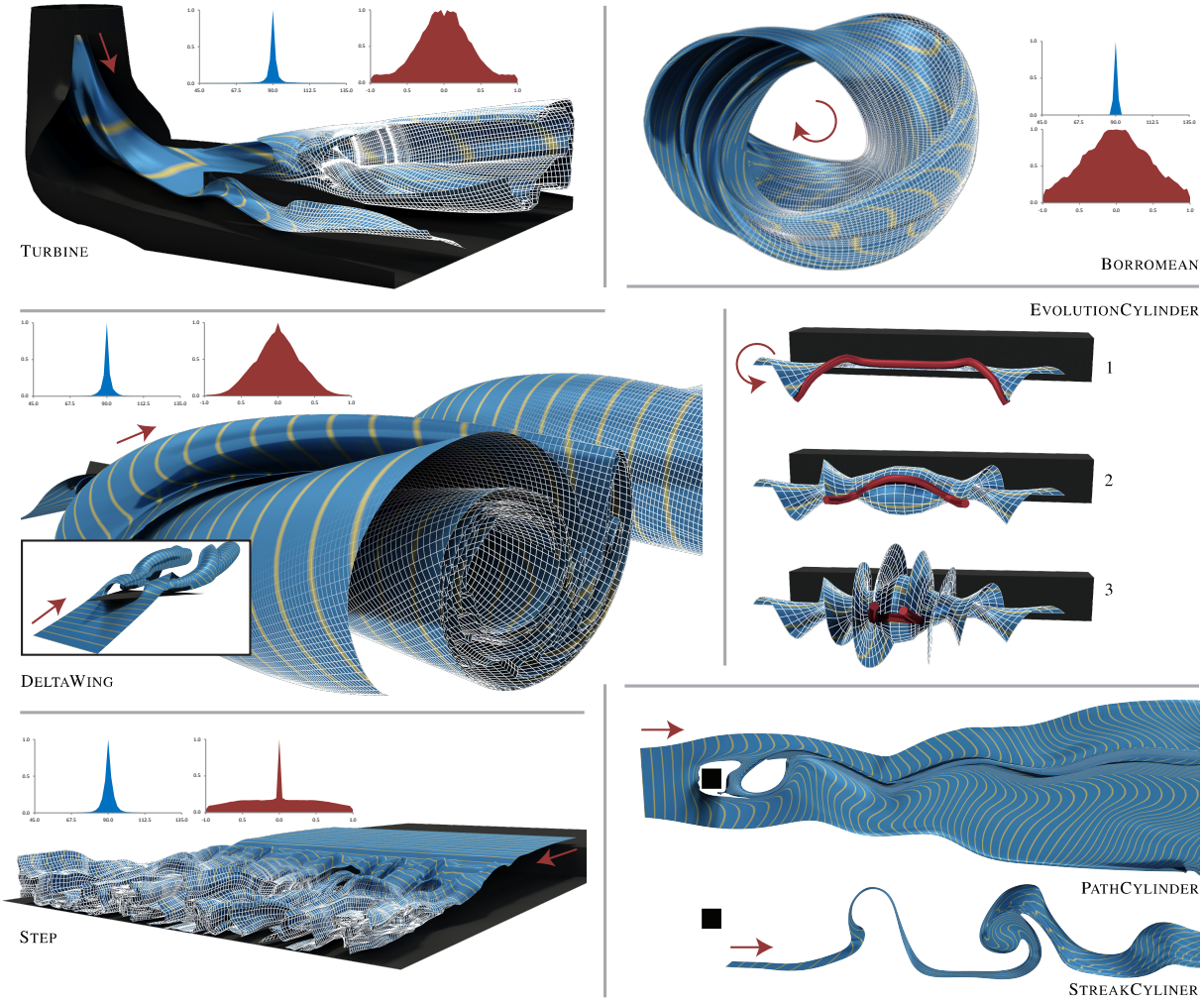

Figure 7: *Results generated by our method for various simulation data. The histograms show quality measures based on angles relative edge lengths (see section 7).* PATHCYLINDER *and* STREAKCYLINDER *show examples for time-dependent vector fields.*

7.2. Simulation Data Sets

We apply our method to a series of more complex, simulated data sets with different properties. The results are shown in Figure 7. For these data, we measure angles between all adjacent edges of the mesh and show their distribution (blue histograms, ideal value is 90°). We also compute the relative deviation from the "ideal" edge length: assuming square elements this is the average length ℓ_{avg} of the four edges. The red histograms show the distribution of $1 - \frac{\ell_i}{\ell_{\text{avg}}}$ for all edge lengths ℓ_i such that values of zero indicate perfect squares.

The TURBINE models the outflow of a hydroelectric turbine. The fluid passes through a pipe (left) and enters a chamber with an obstacle dividing the flow. The stream surface shows that the ripping criterion works reliably for practical data as well. The histograms confirm that quads almost rectangular (blue) and most of them close to square (red).

The time-dependent DELTAWING vector field represents the airflow around a triangle-shaped airplane. The main flow feature are two vortices behind the airplane. These structures are clearly visible in the stream surface that was generated for a single time step. Although the surface is highly twisted it is well tessellated.

The STEP vector field [KJ00] represents the flow behind a backward-facing step, and we examine one time step. The generated stream surface shows that after the flow passes the step, it remains nearly laminar for some time until it suddenly becomes turbulent. The main vortex core region contains many saddles, which all are detected by our ripping criterion. The stream surface is ripped into a couple of "fringed bands". Although the front lines are deformed significantly, the mesh is well tessellated and consists of almost equilateral quads. This data set also shows the robustness of our approach for nontrivial flow behavior.

The time-dependent BORROMEAN [DSCB10] data set represents the decay of a magnetic field. The initial configuration consists of interweaved Borromean rings of magnetic flux. We show a stream surface that is seeded in the vicinity of closed orbits. This example shows that regions of high curvature as well as long integration times are handled well.

The EVOLUTIONCYLINDER data set [CSBI05] shows the evolution of a stream surface in the flow around a square cylinder. The stream surfaces are seeded behind the cylinder. This is the region where a vortex core line appears. We generate a stream surface for one single time step and show three stages of the integration with the front marked red. Note that there is no ripping despite high twisting and shear: the stream surface is always ruled by a single front line.

We use the same data to show that – with few extensions – our algorithm can as well handle time-dependent vector fields, i.e., it can generate path and streak surfaces. The results are shown as PATHCYLINDER and STREAKCYLINDER in Figure 7.

8. Discussion

In this section we discuss the contribution of our approach, relation to other approaches, quality, performance, potential generalizations, and limitations.

Contribution and relation to other approaches. Our work is based on the novel idea of integrating a modified flow $\alpha\mathbf{v}$ instead of $\mathbf{v}$: the scalar function α scales velocity locally such that the front line of a stream surface is aligned orthogonally to the flow. This is essentially a reparametrization of the stream surface that enforces a rather uniform advancement of the front and hence a high quality surface mesh consisting of well-shaped quads. At the same time the velocity scale α provides a reliable and robust detector for critical points. This is the main contribution of this work. In contrast to other approaches, our method benefits from the fact that it is based on a global observation of the flow and the stream surface.

The reparametrization, the core part of our method, is independent of the particular mesh generation. It can be plugged into *any* method for constructing stream surfaces that is based on a *marching front* algorithm. This renders the main contribution of our approach in a sense "orthogonal" to existing approaches. It is easy to extend or to even exchange of the mesh generation by a more sophisticated algorithm without additional effort: integrate the scaled flow $\alpha\mathbf{v}$ and react on detection of critical points, e.g., to close gaps at saddles. Note that a significant part of special treatment required by other methods may not be necessary anymore when following our approach. This includes certain optimizations of the triangulation such as stripification for efficient rendering due to our automatic alignment.

Performance The performance of any method for stream surface integration depends on several parameters that allow for a trade-off between computational cost and quality of the result. From this point of view our approach is similarly efficient as others: the only additional requirement is the repeated computation of the 1-dimensional kernel of a bidiagonal matrix. This can be done in $O(n)$, where n is the number of vertices on the streamline. We implemented a customized solver, which is numerically stable and highly efficient (see Section 5). However, in contrast to other methods, a *parallel* implementation of our approach is straightforward. Also note that our flow-orthogonal integration potentially reduces the computational cost for integration and meshing due to sampling a more suitably parametrized representation of the stream surface. This effect is hard to quantify.

Data set	n_s	n_i	n_v	adapt	α	rip	advance	triangulate
Lorenz	10	500	287,869	1,116	1,539	569	231	683
2D Saddle	20	240	8,821	350	319	142	89	265
3D Saddle	50	300	76,117	322	327	136	85	260
BubbleChamber	50	100	7,251	189	252	80	403	143
Cylinder	100	500	50,021	510	500	217	264	435
Cylinder	200	1,000	199,690	972	962	416	293	8159
Borromean	20	2,000	138,272	2,082	2,126	887	3,014	1,738
Borromean	200	4,000	425,118	3,856	3,766	1,651	763	3,201

Table 1: *Timings for stream surfaces of data sets with n_s vertices on the seed line performing n_i integration steps. We measured time to adapt, rip, compute α, advance the front and, triangulate. All times are measured in milliseconds.*

We applied our algorithm on a series of increasingly complex benchmark data sets, some are shown Section 7. Timings for all data sets are given in Table 1, which shows the typical computation times for the different steps of mesh generation. The measurements show that around 28% of the computation time for one iteration of front advancement are spent for computing α.

Surface Quality It is not an easy task to propose a surface quality measure that includes both accuracy of the stream surface approximation and quality of its parametrization, i.e., quality of the mesh. As the first quantity depends mainly on the numerical integration scheme, we focus on the parametrization and measure quality of the generated quads based on angles and relative edge lengths (see Section 7). The histograms in Figure 7 show that integration in the scaled $\alpha\mathbf{v}$ results in almost perfectly rectangular quads. Moreover, most quads are close to squares. The average portion of quads in the meshes we obtained throughout experimentation is $> 95\%$. The examples in Figure 6 show that existing non-recursive methods [GKT*08] have a significantly lower tessellation quality. In fact, there, the small angle problem leads to long and thin triangles, contrary to our approach.

Generalization to other surfaces We introduced the theoretical concept of our method in Section 3 for stream surfaces in steady flow fields. As already shown with the results (see Figure 7 PATHCYLINDER and STREAKCYLINDER), the concept can be extended to time-dependent vector fields and *path surfaces*. (See, e.g., [MLP*10] for a definition of path and streak surfaces.) This extension is straightforward because the path surface of an unsteady flow $\mathbf{v}(\mathbf{x},t) \in \mathbb{R}^d$

can be described as stream surfaces of the steady $d+1$-dimensional flow $(\mathbf{v},1)^T \in \mathbb{R}^{d+1}$ [TWHS05]. Similarly, *streak surfaces* in $\mathbb{R}^d$ can be represented as stream surfaces in $\mathbb{R}^{d+1}$ [WT10]. The latter, however, requires a rather expensive preprocessing to convert the data set.

Limitations Our approach also has limitations: the above mentioned integration of path surfaces and streak surfaces requires the simultaneous evaluation of flow data from different time steps. This increases the size of the working set and may reduce the maximum spatial data resolution compared to other methods. This issue could be alleviated by an out-of-core implementation, which works only on a narrow band of the flow.

9. Conclusions

We developed a new approach to the generation of stream surfaces that aligns the front line orthogonally to the flow. This is achieved by minimizing an error function at each integration step of the front line. We derived the error function in the continuous setting, and showed that the discretization essentially leads to solving a linear problem. Flow-orthogonal integration can be interpreted as a conformal reparametrization of stream surfaces, which allows for better sampling by the advancing front. This way, we obtain high quality triangles meshes in interactive applications. Our approach is efficient and robust, and it can easily be combined with other methods or existing implementations.

Acknowledgements

We thank Janick Martinez Esturo for his help with the video and experimentation. We also thank Tino Weinkauf for providing a resampled version of the cylinder data set. The delta wing data set is courtesy of Markus Rütten from DLR.

References

[CSBI05] CAMARRI S., SALVETTI M.-V., BUFFONI M., IOLLO A.: Simulation of the three-dimensional flow around a square cylinder between parallel walls at moderate Reynolds numbers. In *XVII Congresso di Meccanica Teorica ed Applicata* (2005). 9

[DH97] DELLNITZ M., HOHMANN A.: A subdivision algorithm for the computation of unstable manifolds and global attractors. *Numerische Mathematik 75*, 3 (1997), 293–317. 3

[DKK91] DOEDEL E. J., KELLER H. B., KERNÉVEZ J. P.: Numerical analysis and control of bifurcation problems. I. Bifurcation in finite dimensions. *IJBC 1*, 3 (1991), 493–520. 3

[DSCB10] DEL SORDO F., CANDELARESI S., BRANDENBURG A.: Magnetic-field decay of three interlocked flux rings with zero linking number. *Phys. Rev. E 81*, 3 (2010), 36401 – 36408. 9

[GKT*08] GARTH C., KRISHNAN H., TRICOCHE X., TRICOCHE T., JOY K. I.: Generation of accurate integral surfaces in time-dependent vector fields. *IEEE TVCG 14*, 6 (2008), 1404–1411. 2, 6, 7, 9

[GTS*04] GARTH C., TRICOCHE X., SALZBRUNN T., BOBACH T., SCHEUERMANN G.: Surface techniques for vortex visualization. In *Proc. VisSym* (2004), pp. 155–164, 346. 2

[GV04] GUCKENHEIMER J., VLADIMIRSKY A.: A fast method for approximating invariant manifolds. *SIADS 3*, 3 (2004), 232–260. 3

[Hen05] HENDERSON M. E.: Computing invariant manifolds by integrating fat trajectories. *SIADS 4*, 4 (2005), 832–882. 3

[Hul92] HULTQUIST J. P.: Constructing stream surfaces in steady 3-d vector fields. In *Proc. IEEE Vis* (1992), pp. 171–178. 2, 3, 6

[KJ00] KALTENBACH H.-J., JANKE G.: Direct numerical simulation of flow separation behind a swept, rearward-facing step at re[sub h] = 3000. *Physics of Fluids 12*, 9 (2000), 2320–2337. 8

[KO99] KRAUSKOPF B., OSINGA H.: Two-dimensional global manifolds of vector fields. *CHAOS 9* (1999), 768–774. 3

[KO03] KRAUSKOPF B., OSINGA H. M.: Computing geodesic level sets on global (un)stable manifolds of vector fields. *SIADS 2* (2003), 546–569. 3

[KOD*05] KRAUSKOPF B., OSINGA H. M., DOEDEL E. J., HENDERSON M. E., GUCKENHEIMER J., VLADIMIRSKY A., DELLNITZ M., JUNGE O.: A survey of methods for computing (un)stable manifolds of vector fields. *IJBC 15*, 3 (2005), 763–791. 3

[MLP*10] MCLOUGHLIN T., LARAMEE R. S., PEIKERT R., POST F. H., CHEN M.: Over two decades of integration-based, geometric flow visualization. *Computer Graphics Forum 29*, 6 (2010), 1807–1829. 9

[MLZ09] MCLOUGHLIN T., LARAMEE R. S., ZHANG E.: Easy integral surfaces: a fast, quad-based stream and path surface algorithm. In *Proc. CGI* (2009), pp. 73–82. 3

[PS09] PEIKERT R., SADLO F.: Topologically Relevant Stream Surfaces for Flow Visualization. In *Proc. SCCG* (2009), pp. 43–50. 3

[SBH*01] SCHEUERMANN G., BOBACH T., HAGEN H., MAHROUS K., HAMANN B., JOY K. I., KOLLMANN W.: A tetrahedra-based stream surface algorithm. In *Proc. IEEE Vis* (2001), pp. 83–91. 2, 6

[SRGT12] SCHULZE M., RÖSSL C., GERMER T., THEISEL H.: As-perpendicular-as-possible surfaces for flow visualization. In *Proc. IEEE PacificVis* (2012), pp. 153–160. 3

[SRWS10] SCHNEIDER D., REICH W., WIEBEL A., SCHEUERMANN G.: Topology aware stream surfaces. *CGF 29*, 3 (2010), 1153–1161. 2, 3, 6

[SST*07] SCHAFHITZEL T., STUTTGART U., TEJADA E., WEISKOPF D., ERTL T.: Point-based stream surfaces and path surfaces. In *Proc. GI* (2007), pp. 289–296. 3

[Sta98] STALLING D.: *Fast Texture-Based Algorithms for Vector Field Visualization*. PhD thesis, FU Berlin, 1998. 2, 3, 6, 7

[SWS09] SCHNEIDER D., WIEBEL A., SCHEUERMANN G.: Smooth stream surfaces of fourth order precision. *CGF 28*, 3 (2009), 871–878. 2

[TSW*05] THEISEL H., SAHNER J., WEINKAUF T., HEGE H.-C., SEIDEL H.-P.: Extraction of parallel vector surfaces in 3d time-dependent fields and applications to vortex core line tracking. In *Proc. IEEE Vis* (2005). 5

[TWHS05] THEISEL H., WEINKAUF T., HEGE H.-C., SEIDEL H.-P.: Topological methods for 2d time-dependent vector fields based on stream lines and path lines. *IEEE TVCG 11*, 4 (2005), 383–394. 10

[WT10] WEINKAUF T., THEISEL H.: Streak lines as tangent curves of a derived vector field. *IEEE TVCG 16*, 6 (2010), 1225–1234. 10