Generation of Accurate Integral Surfaces in Time-Dependent Vector Fields

Christoph Garth, Hari Krishnan *Student Member, IEEE*, Xavier Tricoche, *Member, IEEE*, Tom Bobach *Student Member, IEEE* and Kenneth I. Joy, *Member, IEEE*

Abstract—We present a novel approach for the direct computation of integral surfaces in general vector fields. As opposed to previous work, which we analyze in detail, our approach is based on a separation of integral surface computation into two stages: surface approximation and generation of a graphical representation. This allows us to overcome several limitations of previous techniques. We first describe an algorithm for surface integration that approximates a series of timelines using iterative refinement and computes a skeleton of the integral surface. In a second step, we generate a well-conditioned triangulation. The presented approach allows a highly accurate treatment of very large time-varying vector fields in an efficient, streaming fashion. We examine the properties of the presented methods on several example datasets and perform a numerical study of its correctness and accuracy. Finally, we examine some visualization aspects of integral surfaces.

Index Terms—3D vector field visualization, flow visualization, time-varying and time-series visualization, surface extraction

1 Introduction

Integral surfaces, as a natural generalization of integral lines, are a basic yet powerful tool for insightful vector field visualization. Representing a continuum of integral lines, they constitute a surface and allow for the application of surface shading techniques. Hence, in comparison to streamlines or other line-based techniques, they support depth perception and greatly facilitate the visual understanding of complex three-dimensional structures.

Integral surfaces appear quite naturally in many problems that involve vector fields; the most prominent example is *flow visualization* that centers on the visual analysis of datasets generated by simulation (Computational Fluid Dynamics) or measurement. In the context of flows, the groundbreaking drawing work of Dallmann [3] has shown that many types of flow patterns may be well understood in terms of so-called *flow sheets*. These sheets represent precisely chosen integral surfaces that emanate from specific lines on the surface of objects embedded in a flow, and often take on the role of flow separators that divide regions of differing behavior. In the broader context of dynamical systems, integral surfaces appear as separatrices of three-dimensional vector fields, where they represent the two-dimensional stable or unstable manifolds associated with critical points. However, even if not coupled to such specific interpretations, integral surfaces have been shown to possess great illustrative power (cf. [2, 4, 8]). In the case of flows, they can accurately depict folding, shearing and twisting and impart an intuitive understanding of the flow geometry.

In recent years, several approaches have been presented for the computation and graphical representation of integral surfaces in CFD datasets. However, with the notable exception of [14], this work has focused on the computation of *stream surfaces*, i.e. integral surfaces in steady vector fields, as opposed to *path surfaces*, which denote the time-dependent case. Much of this prior work has concentrated on the generation of a viable graphical representation using advancing front methods, based on adaptive refinement heuristics that are aimed at producing good surface meshes. However, the accuracy of the resulting surface, which is a crucial property for some application scenarios, has not been examined in detail. This is in part attributed to the generally complex form of the resulting algorithms that make such discussion difficult.

In this work, we present a novel algorithm for the computation of integral surfaces in a general context. Our method is based on the adaptively-refined advancing front paradigm, and is applicable to both stationary and time-varying vector fields. Treatment of the latter is achieved in a streaming fashion, thus allowing our method to work even on extremely large datasets with thousands of time steps.

After reviewing the basic concepts underlying integral surfaces and fixing terminology in Section 2, we review previous work and perform an analysis of its limitations in Section 3. We then proceed to develop a generic two-step integral surface approximation scheme in Section 4, and discuss practical issues of its application in Section 5. Finally, in Section 6 we briefly examine some visualization aspects of integral surfaces, before we conclude on the presented material in Section 7.

2 Concepts

In the following, we assume that v is a three-dimensional vector field, defined over a domain $\Omega \subset R^3$ and a time interval $[T_0, T_1]$. An *integral curve* $S(t_0,x_0;t)$ of v is the solution to the ordinary differential equation

$$\left.\frac{d}{dt}S(t_0,x_0;t)\right|_{t=\tau} = v(\tau, S(t_0,x_0;\tau)). \tag{1}$$

Simply put, it is a curve that contains the point (t_0,x_0) and is tangent to the vector field at every point over time. The intuitive understanding associated with such integral curves is that of massless particles that are advected through a domain by a vector field with time. For the class of Lipschitz-continuous vector fields, existence and uniqueness of S can be proven (cf. [1, 6]), and numerical integration methods can be used to approximate the solution. Note that for the overwhelming majority of application datasets, this condition holds true.

Now, let C be a space curve, contained in Ω and parameterized by s, and $T_0 < t_0 < T_1$. Then, an *integral surface* I is defined by

$$I(s,t) \;:=\; S(t_0, C(s); t).$$

In words, I is the union or continuum of integral curves passing through the *seeding curve* C at time t_0. While $I(s,\cdot)$ coincides with an individual integral curve, the surface lines given by $I(\cdot,t)$ are called *time lines*. Figure 1 provides a more graphical explanation of these terms.

- *C. Garth, H. Krishnan and K. I. Joy are with the Institute of Data Analysis and Visualization, University of California, Davis, E-mail: {cgarth,hkrishnan,kijoy}@ucdavis.edu.*
- *X. Tricoche is with the Computer Science Dept., Purdue University, E-mail: xmt@cs.purdue.edu.*
- *Tom Bobach is with the Geometric Algorithms Group, University of Kaiserslautern, E-mail: bobach@informatik.uni-kl.de.*

Manuscript received 31 March 2007; accepted 1 August 2007; posted online 27 October 2007.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

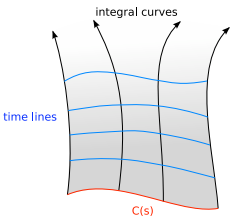


Fig. 1. Schematic overview of integral surface constituents.

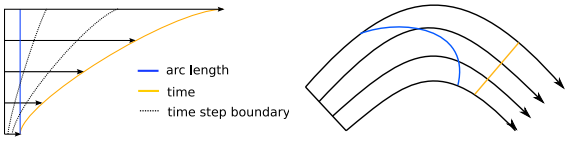


Fig. 2. Arc length and time parameterization of integral curves can diverge over time and span multiple time steps.

Observe that I has a natural parameterization in the form of (s,t)-coordinates. Every point on the surface, given its parametric coordinates (s,t), can be computed by propagating the unique streamline starting at $C(s)$ through the application of a numerical integration method until it reaches t. The goal of any integral surface algorithm is to construct a geometric approximation I using a sparse set of integral curves.

Remark that I (and in turn its time lines) need be continuous in s, e.g. if the surface encounters a boundary of Ω. In this case, the corresponding integral curve cannot be continued. Similar behavior is induced by critical points in the flow; while those integral curves converging to a critical point take infinite time to reach it, other parts of the surface may flow past and diverge strongly. We refer to both these phenomena as *tearing*, and correct approximation of must be treated with some care. In practical applications, it is safe to assume that an integral surface exhibits only a finite number of such occurences.

In the next section, we proceed to discuss several approaches that have been proposed in previous work.

3 Previous Work

In the following, we give a brief overview of existing techniques for integral surface computation, and analyze specific shortcomings that preclude their application to large time-varying datasets.

3.1 An Overview of Computation Techniques

The research on efficient generation of stream surfaces was essentially started by Hultquist in [8]. He was the first to describe surface integration in terms of an advancing front. In his approach, the starting curve is discretized through a finite set of points, each of which is used to seed a streamline integration, forming a skeleton of the stream surface. The streamlines are advanced a single (fixed) integration step at a time, resulting in a sequence of points for each streamline, and the advancing front is defined as the polyline connecting the current streamline points. Each adjacent pair of streamlines form a *ribbon*, and the ribbon contains a front segment connecting two streamline points in a line segment. This segment is advanced by considering the two diagonals formed by the current front points and the next integration points on both streamlines, and the shorter diagonal is selected as the new front segment. Then, to ensure that the front remains continuous, adjacent ribbons are advanced in turn until the front is consistent once more. This process is repeated until all streamlines have reached their maximum integration length. During the front advancement, the triangles spanned between old and new front segments are stored in a list and make up the final geometric representation of the surface.

Adaptive front refinement was handled by Hultquist in two ways: First, to adapt the front resolution to stretching of the stream surface, he proposed a simple criterion based on the distance between adjacent streamlines; if it exceeds a user-prescribed threshold, a new streamline is inserted in the middle of the current front segment, and the corresponding ribbon is split in two. Furthermore, a threshold criterion is applied on the direction of neighboring streamlines, and if they appear to move in opposing directions, the ribbon is terminated. This criterion reflects the fact that stream surfaces may become discontinuous at or near saddle-like structures. This method performs well for moderately complex flows, is however limited in efficiency by the use of fixed step-size integration. Furthermore, it does not cope well with folding, shearing or twisting of stream surfaces, since the refinement strategy does not take these issues into account.

Hultquist's algorithm was augmented by Stalling in [17] by incorporating local topological information into the triangulation process and slightly modifying the refinement criteria to address some of the cases mentioned above. For the case of tetrahedral grids, Scheuermann et al.[15] exploited the existence of an analytic flow solution for tetrahedral grids endowed with linear interpolation to compute a stream surface on a per-tetrahedron basis. Due to the linear nature of the vector field inside every grid cell, streamline paths are available in closed form. While this technique is conceptually elegant, the resolution of the resulting surface is closely tied to that of the underlying mesh, and the limitation to vector fields given on tetrahedral meshes is a serious impediment to the treatment of application cases that base model computations on other interpolation schemes.

Most recently, Garth et al. [4] introduced several modifications to Hultquist's original scheme to allow stream surface computation in the presence of very complex flow structures. By moving from time-based streamline integration to arc length-based integration and incorporating front refinement criteria based on curvature, better resolution control is achieved, and complex flow patterns can be adequately resolved.

In contrast to these methods that compute the surface geometry explicitly as a triangle mesh, van Wijk[18] gave a global approach that implicitly represents a family of stream surfaces. Through advection of a scalar field from the domain boundary through the flow domain, the computation of a particular stream surface is transformed into the extraction of an isosurface. The latter problem is well understood, and many reliable methods are available (e.g. [12]). The generation of the scalar field itself, however, requires extensive processing. Furthermore, selecting stream surfaces is very non-intuitive, and defining a suitable boundary field is hard. Cai and Heng[2] have shown that for a very limited class of flows, so-called *principal stream surfaces* can automatically select and compute interesting surfaces.

Aside from explicit construction of stream surface geometry, methods exist that create the visual impression of a stream surface by using particles, e.g. [19]. While simple to implement and broadly applicable, the visual clarity of these schemes is often lacking, as the depth-enhancing quality of shaded surfaces is lost. More recently, Schafhitzel et al. [14] described an approach for both stream surfaces and path surfaces that relies on the massive integration of particles on the GPU and renders the resulting point set through a splatting method. While this method offers fast surface integration, it was demonstrated only for regular meshes, and its suitability for large unstructured datasets is uncertain. Additionally, by not obtaining an explicit representation of the surface geometry, point-based methods forfeit the use of integral surfaces as building blocks for advanced visualization techniques (e.g. [4]).

3.2 Analysis

In practice, Hultquist's method and its derivatives are most commonly used. This can be attributed to the fact that they offer reasonable efficiency and applicability to moderately complex flows. However, there are a number of drawbacks to this class of algorithms.

First and foremost, they do not generalize efficiently to time-dependent settings, i.e., they do not allow the computation of path surfaces in large time-varying datasets. An analysis of the original method shows that in order to achieve a viable triangulation of the surface, the front advances non-uniformly with time. Therefore, a sin-

gle front advancement pass must potentially make use of many time steps simultaneously to achieve the necessary temporal interpolation required for the propagation of the individual front pathlines. While it is theoretically possible to compute many pathlines in a single pass over all time steps, a reformulation of the algorithm along these lines is not efficient since the adaptive refinement mandates the insertion of new pathlines during the front advancement, requiring (potentially arbitrarily many) further passes over the dataset. The same drawback can be observed on the arc length-based integration scheme from [4], as the varying velocity magnitude of a vector field induces a divergence between the time- and arc length-parametrizations of an integral curve. Figure 2 illustrates this effect for both schemes.

Secondly, the implementation complexity of Hultquist-type schemes is enormous. The combination of iterative advancement of single front segments and the recursive advancement of neighboring ribbons poses a significant obstacle to the incorporation of different refinement criteria. While refinement based on a pair of adjacent integral curves is straightforward, incorporating more complex criteria such as curvature (which must be computed from a triple of neighboring front points) requires a reformulation of the algorithm in a purely iterative shape. It is our experience that such implementations are highly complex and can be quite fragile with respect to the ordering of different refinement criteria.

Third, the use of fixed step sizes in the numerical integration of the underlying integral curves precludes the use of integration techniques that make use of adaptive step size control, such as the very popular Runge-Kutta-Fehlberg method (cf. [7]). Therefore, the step size parameter is subject to a choice between the accuracy of the surface approximation and the invested computational effort. For complex vector fields, this typically results in either a too coarsely resolved surface that may exhibit visual artefacts and is likely incorrect, or a very finely resolved surface with many millions of triangles that is hard to depict at interactive speeds and requires extended computation times.

The root cause of many of the outlined problems is to be found in a lack of separation of surface approximation on one hand and the generation of a representative mesh that is suitable for rendering on the other. In this work, we address the outlined issues by effectively providing separate algorithms for both of these tasks.

Last, we wish to point out that the correctness and accuracy of integral surface algorithms have not been systematically examined in the past. As the advancing front is approximated by a curve segment (typically a line segment or a spline), inserting new integral curves on the midpoint of such a segment leads to a small error, since its initial position is not an actual surface point. This error is propagated throughout the integration and can, in the worst case, result in a surface geometry that does not adhere to the vector field under consideration. This problem is especially aggravated if the surface inherits its interpretation in terms of the application from the seeding curve, as is the case for e.g. surface or saddle separation surfaces. Hence it is critical to assess the qualitative behavior of the overall surface algorithm and specifically the refinement strategies.

3.3 Integral Surface Visualization

Regarding the topic of visualization, several visualization techniques have been proposed beyond direct display of the surface. Löffelmann et al. [11] introduced the concept of *stream arrows*, i.e. a specific texture map on stream surfaces that generates arrow-shaped holes in the surface to address occlusion problems and indicate flow direction. Garth et al. [4] examined the use of stream surfaces as a generalization of slicing planes and employed color mapping to visualize derived flow quantities on such surfaces. In the same work, stream surfaces were also applied as a building block for a vortex extraction scheme. More recently, the application of texture-based flow visualization techniques on stream surfaces in the form of *Image-Space Advection* was demonstrated by Laramee et al. [10], and recently generalized to *Line Integral Convolution* on point-based path surfaces by Schafhitzel et al. [14].

4 Direct Approximation of Integral Surfaces

In the following, we present a novel technique for the approximation of integral surfaces in general settings. This approximation is achieved by the successive approximation of timelines. We first describe a generic curve refinement algorithm and then use it to construct the surface approximation.

4.1 Generic Curve Refinement

Let $f : [s_{\min}, s_{\max}] \to R^3$ a piecewise smooth curve with at most finitely many points of discontinuity, and (s_i) a monotonic sequence of parameters covering $[s_{\min}, s_{\max}]$. Denote by $f_i = f(s_i)$ the corresponding sequence of curve points. Then, the piecewise linear interpolant Lf of the f_i is an approximation of f.

We next describe a generic algorithm for the adaptive refinement of this approximation by incremental insertion of new parameters into the sequence of parameters (s_i). To achieve this in the presence of discontinuities, we introduce two predicates

$$Q_{\text{discont}} : (s_{i-1}, f_{i-1}), (s_i, f_i), (s_{i+1}, f_{i+1}) \;\mapsto\; \{\text{true}, \text{false}\}$$

and

$$Q_{\text{refine}} : (s_{i-1}, f_{i-1}), (s_i, f_i), (s_{i+1}, f_{i+1}) \;\mapsto\; \{\text{true}, \text{false}\}.$$

that estimate the properties of the approximation over a triplet of adjacent parameters/curve points. The first predicate responds true if a discontinuity of first or second order is estimated from the current sequence of points, and the second predicate indicates where the approximation quality may be insufficient and new parameters should be inserted to generate a better approximation.

We formulate the following iterative refinement algorithm, based on midpoint insertion:

1. Let (s_i) and (f_i), $i = 0, \ldots, M$ an (externally provided) initial approximation, and let $S_{\text{insert}} = \emptyset$.

2. For each triplet of adjacent points (s_{i-1}, f_{i-1}), (s_i, f_i), (s_{i+1}, f_{i+1}) where Q_{discont} evaluated true, split the interval $[s_{\min}, s_{\max}]$ into two subintervals $[s_{\min}, s_i]$ and $[s_{i+1}, s_{\max}]$, and recurse to step 1 for each of them.

3. For each triplet of consecutive points (s_{i-1}, f_{i-1}), (s_i, f_i), (s_{i+1}, f_{i+1}) where Q_{refine} evaluates true, insert $\frac{1}{2}(s_i + s_{i+1})$ and $\frac{1}{2}(s_{i-1} + s_i)$ into S_{insert}.

4. If $S_{\text{insert}} = \emptyset$, finish.

5. Compute the sequence F_{insert} by evaluating f at the parameters in S_{insert}, and merge the S_{insert} and F_{insert} into the sequences (s_i) and (f_i), respectively. Continue at step 2.

Depending on the choice of predicates, this algorithm refines the parameter sequence until a certain level of quality, as measured by the predicates, is reached. If a point of discontinuity is detected, the problem is split into two subproblems, which are then recursively refined. If no a priori information about f is available, a uniform sequence of (s_i) of a minimum resolution is a good choice.

Remark that in the above description, since the predicate Q_refine is evaluated over two adjacent segments, and without additional information. Hence, we must insert both interval midpoints into (s_i), as inserting either midpoint might not improve the curve discretization. While limited choices of Q_refine allow a reformulation of the above algorithm in terms of point pairs or single segments, we opt for the above more general description in the interest of uniform presentation.

The present algorithm is built on the requirement that after a finite number of refinement iterations, Q_{refine} will evaluate false for every triplet of nodes. To guarantee this, we limit our choice of predicates to the thresholding of curve properties, derived from triplets of sequence points, that converge to zero as a smooth curve is increasingly refined.

Typical choices that fulfill this requirement are distance between successive f_i, second differences of f_i, inter-segment angle, triplet triangle area, or any combination of these.

Conversely, the presence of discontinuities of f will inhibit the convergence of such criteria. Therefore, Q_{discont} must serve as a robust estimator of curve discontinuities, such that refinement is not attempted across the discontinuity and proceeds only over the smooth regions of f. A good choice is a large upper bound on the magnitude of the first derivative and second derivative of f, as approximated by the finite differences over point triplets.

Given such a choice of predicates, the refinement algorithm is very robust, even in the presence of numerical errors. In the next section, we will apply this algorithm to the incremental refinement of time lines.

4.2 Successive Time Line Approximation

We now formulate integral surface computation in terms of successive approximation of time lines. In the following, the notations of Section 2 apply.

Let (t^n), $n = 0,\ldots,N$ a sequence of time parameters covering the interval $[T_0, T_1]$, and we assume that the time line I^n is given as a finite number of linear or smooth segments. Then, the next time line

$$I^{n+1}(s) := S(t^n, LI^n(s)\,; t^{n+1})$$

is approximated using the algorithm given above, where again L denotes the piecewise linear interpolant of I^n,

The sequence is started by letting $I^0 := C$ and terminated once I^n is approximated. Remark that, in order to approximate the time line I^{n+1}, instead of performing refinement with a uniform parameter sequence, we reuse the sequence of parameters of I^n. This guarantees that the time lines are increasingly refined as their complexity increases. Hence, the set of parameters of I^{n+1} is always a superset of the parameters of I^n.

Since timeline points result from solutions of the ordinary differential equation (1 and v is assumed Lipschitz-continuous, *local* approximation errors between successive time lines I^n and I^{n+1} are bounded. Hence, the accumulated, *global* approximation error for I^N is also bounded and a function of the local error. This ensures that correctness and accuracy of integral surfaces is achievable by limiting the local error. Unfortunately, except for the simplest choices of v, it is not possible to quantify the above statement. We examine this issue in more detail through numerical experiment in Section 5.2.

Before we proceed to discuss more practical aspects of integral surface approximation, we wish to point out two important properties of the above scheme. First, it is local in time, in the sense that to approximate I^{n+1} from I^n, the algorithm makes use of the values of v only over the interval $[t^n, t^{n+1}]$. Second, once I^{n+1} has been approximated, the set of integral curves corresponding to the set of parameters of I^{n+1} after refinement form a skeleton of the surface over the interval $[t^n, t^{n+1}]$.

4.3 Efficient Surface Computation

The above surface approximation scheme is built on the integral curves propagating along the sequence of time lines, and they must be approximated through numerical integration.

Typically, for each of these curves, a chosen numerical integration scheme outputs a sequence of points, according to either fixed or adaptive choices of step size, that are then used to represent the integral curve in piecewise linear fashion. If graphical requirements must be met, such as an upper bound on the angle of successive line segments, the step size must be forcibly chosen to fulfill these requirements, or the curve must be resampled. This in general stands in the way of maximum efficiency and accuracy of the numerical computation of integral curves. In order to avoid this limitation, we make use of a certain class of integration schemes with *dense output*, for which the output is not a sequence of points but rather a sequence of polynomials, typically globally at least C^1-continuous and provably convergent to the actual solution (we refer the reader to e.g. [5, 16] for an in-depth discussion). For the specific class of Runge-Kutta methods that are typically used for vector field visualization, these piecewise interpolants can be constructed directly from the Runge-Kutta stages, and thus incur no further computational cost. We make use of the fourth-order adaptive DOPRI5 integration scheme described by Prince and Dormand in [13]. It is then a simple matter to postpone the generation of a discretization of each integral curve for graphical purposes until it is actually needed. Hence, the integration process can proceed with maximum efficiency over the time intervals $[t^n, t^{n+1}]$ as discussed above.

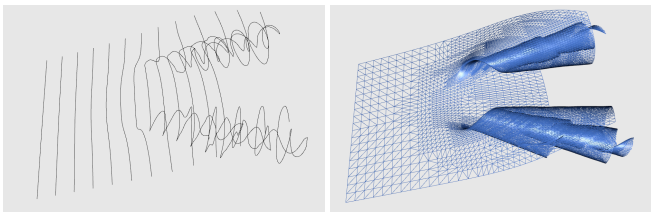


Fig. 3. Overview of the integral surface computation through successive time line approximation (top image) and resulting surface triangulation (bottom image) in the Delta Wing dataset.

In practical applications, v may contain critical points or object boundaries, hence integral lines may not continue from I^n to the next time line I^{n+1}. Such cases, as well as integral curves converging to a critical point, are indicated by an error condition of the underlying numerical scheme. Owing to the robustness of the described refinement scheme, we can still perform time line approximation in an unchanged way in these cases. Then, to approximate I^{n+2}, we remove the intervals containing terminated integral curves from the initial approximation of I^{n+2} and proceed with the refinement process.

Any discontinuities of first or second order introduced by this procedure into the time line are automatically handled by the refinement scheme, and the timeline is refined until the accuracy estimate is met. Besides conceptual simplicity, this approach has two further advantages. First, the intersection curves of the integral surface with object boundaries are resolved according to the refinement criteria, and thus artifacts are avoided near boundaries. Second, the overall determinism and error analysis of the surface approximation process remains valid, and accuracy of the resulting surface is not adversely impacted by the presence of boundaries.

Note that in comparison to Hultquist's stream surface algorithm, we do not attempt to remove points from the advancing front (coarsening), but instead only perform refinement. This is based on the following observations. First, in any but specifically chosen analytic cases, the complexity of the front usually increases as it is advected through the flow. It is typically heavily deformed, but cases where it flattens out in a nice manner are very unlikely in practice. Hence, there is typically very little opportunity for coarsening. Second, accuracy of the resulting surface can suffer greatly from overeager coarsening, and parameters required to steer the coarsening process add an additional layer of complexity and make analysis of the surface approximation algorithm much harder. In our opinion, the minimal gain achievable through coarsening does not warrant the additional complexity.

By performing integral surface approximation as outlined above, we have effectively achieved a decoupling of the approximation through a sequence of time lines and connecting integral curves on one hand, and the generation of a graphical representation on the other. We next turn our attention to the latter problem.

4.4 Graphical Representation

After all time lines have been iteratively approximated, we obtain a skeleton of the surface, consisting of a number of piecewise polynomial integral curves. Each of these curves corresponds to a unique s-parameter of the canonical integral surface parameterization. To find a good discrete approximation of these curves, we apply the iterative refinement scheme of Section 4.1 to each integral curve to generate a sequence of time values t_s^k for the specific purpose of graphical rep-

resentation of the integral curve corresponding to s, and we choose the sequence of time values given by the numerical integral as an initial discretization. In other words, integral curves are refined with the same predicates that were before used for time lines.

We then generate a triangulation from this set of integral curves by grouping them into ribbons, and triangulating each ribbon using a shortest diagonal approach.

More specifically, we begin with partitioning the set of all integral curves into two disjoint sets N_A and N_I, where N_A (*active set*) contains all integral curves that start at T_0, and N_I (*inactive set*) contains the remainder. Then for each pair of integral curves S_0 and S_1 in N_A that belong to an adjacent pair s_0 and s_1 of s-parameters, we choose an *active edge* consisting of the first two points of the integral curves. Then, we examine the two diagonals connecting the current edge points and next points on both curves, and take the shorter one as new active edge, simultaneously generating a triangle formed by the old and new active edge points. Before the active edge is advanced, we determine whether the new active edge will cross a timeline, in which case we must decide if the ribbon needs to be split. This is determined by looking at the subset of inactive integral curves in N_I with s-parameters spanned by the ribbon; If the starting time of one of these is crossed by the active edge, the ribbon is split into a number of ribbons corresponding to the number of crossed integral curves, and the corresponding curves are removed from N_I. Consequently, a merging triangulation is created, and each of the ribbons resulting from the split is triangulated recursively. The algorithm terminates if the active edge moves beyond the last point of either integral curve.

This algorithm generates a crack-free triangle-based representation of the integral surface approximation obtained Section 4.1, which can then be directly rendered to achieve a visual representation of the approximated integral surface. Figure 3 depicts both the original sequence of time lines and the triangle mesh generated from it. The triangulation is directly derived from the timeline approximations.

Note that while it is possible to post-process the generated triangle mesh to reduce its size or generate level-of-detail representations for faster rendering, we have not explored these issues here due to limited space.

After having presented our algorithm in a more theoretical fashion above, we now turn to its application on concrete application datasets.

5 Practical Consideration

In the following, we will describe the application of our integral surface approximation scheme to compute stream surfaces and path surfaces in application datasets. The implementation of the algorithms presented above is easily achieved if facilities for interpolation in large datasets are available.

5.1 Choice of parameters

The successive time line approximation that is the core of our integral surface algorithm requires as input a number of parameters. We adopt the philosophy that the selection of parameters should first and foremost accomplish correct and accurate integration of the surface and address efficiency as a secondary concern.

First, a sequence of time parameters must be supplied which determines at which points in time a time line is approximated. In the case of a discretely represented time-varying vector field, the time values of individual time steps that cover the interval $[T_{\min}, T_{\max}]$ are a natural choice, and we have adopted it in our experiments. In the case of a stationary or analytically described vector field, however, no such a priori choice exists, and we generally use an equally spaced sequence of time steps over $[T_{\min}, T_{\max}]$ that is determined by the user through the specification of a Δt parameter that reflects the dataset characteristics. Usually, such parameters are available by considering the context in which the dataset was generated or described. If Δt is chosen much smaller than optimum, algorithm efficiency may suffer since increased effort is required for the numerical integration of the surface skeleton; however, in our experience such a choice does not negatively influence the accuracy of the resulting surface. On the other hand, if Δt is much larger than required, time line refinement may not occur frequently enough to detect the necessity of refinement, and the resulting surface may not be correct. By using the simple heuristic of examining the average number of integration steps that the DOPRI5 scheme takes over the interval $[t^n, t^{n+1}] = [t^n, t^n + \Delta t]$, it is possible to check whether the time step may be much larger than desirable. In our case, if the average number of steps per integral curve exceeds 10, we halve the time step and retry. This simple logic has worked very well in our experiments.

At first glance more difficult is the selection of adequate refinement predicates. While underrefinement can lead to incorrect surfaces, overrefinement is equally undesirable. As discussed in Section 4.1, we have in this work considered threshold predicates on curve properties derived from time line point triplets. More specifically, in the notation of Section 4.1, we have examined the following quantities:

inter-node distance $\delta := ||f_{i+1} - f_{i-1}||$

inter-segment angle $\gamma := \angle(f_{i-1}, fi, f_{i+1})$

triangle area $A := \frac{1}{2}|(f_i - f_{i-1}) \times (f_{i+1} - fi)|$

The refinement predicate Q_{refine} is then given in terms of one or more of these quality criteria and associated thresholds. If any of these thresholds is exceeded, the predicate evaluates true. While δ basically indicates refinement in regions of time line strecthing, and hence responds to a large derivative along the time line, γ is essentially a measure of time line curvature. The triangle area A combines aspects of both derivative and curvature. While all of these refinement criteria are used successfully in various applications of curve refinement, the empirical nature of vector field datasets and the strongly non-uniform shape of time lines does not permit a clear statement or analytical examination to determine which of these criteria is best chosen for time line refinement. We have therefore turned to numerical experiment to examine these criteria more closely.

5.2 Numerical Experiments

To verify the correctness of our refinement scheme and to be able to compare different refinement criteria in terms of correctness and accuracy, we perform the following numerical experiment. Given a dataset and a seeding curve C and a time interval $[T_{\min}, T_{\max}]$, we first compute a highly resolved time line $\tilde{I}^N$ by propagating a very large number K of integral curves directly from the seeding curve. These integral curves are seeded equidistantly with respect to s on C. We interpret the time line spanned by these integral curve endpoints as the ground truth against which to compare the results of our method. Then, for each experiment, we select a fixed sequence of time parameters (t^n), a fixed quality criterion from the list given in the previous section, and a sequence of progressively lower thresholds that should require increasing refinement. We then compute the sequence of time lines as outlined above, resulting in the final timeline I^N. To measure the approximation quality of our algorithm, we then compare $\tilde{I}^N$ and I^N using the error measures

$$\mathrm{E}_\infty := \max_{k=1,\ldots,K} \left|\left| LI^N(s_k) - \tilde{I}^N(s_k) \right|\right|$$

and

$$\mathrm{E}_2 := \frac{1}{K} \sum_{k=1,\ldots,K} \left|\left| LI^N(s_k) - \tilde{I}^N(s_k) \right|\right|^2.$$

Figure 4 (a)-(c) illustrates the results of these experiments for the case of the time-varying Ellipsoid dataset (see Section 6 for a description of this dataset). It is fairly smooth, but contains many time steps and therefore allows integral surfaces that span a large time interval. We observe that decreasing area and distance thresholds lead to increased refinement and a decrease in overall error, while the angle criterion does not induce a significant reduction in error. The distance criterion apparently allows for a quicker reduction in error (Figures 4(a)

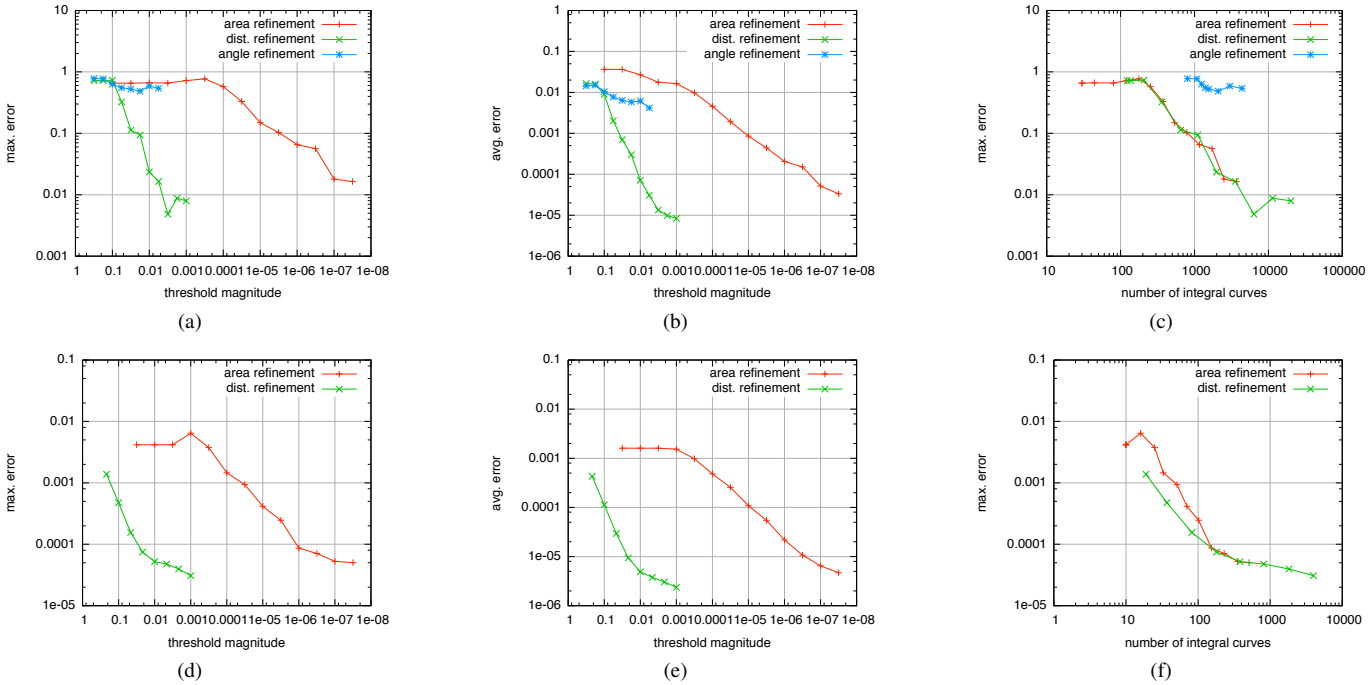


Fig. 4. Results of numerical experiment in the Ellipsoid (top row) and Delta Wing (bottom row) datasets. Note that for the Delta Wing results, the angle refinement failed to produce approximate surfaces and hence no results are shown.

and 4(b)), a plot of error against the number of integral curves (Figure 4(c)) demonstrates that distance and area criteria perform equally well, and we conclude that the area criterion offers more gradual control of refinement.

Figure 4 (d)-(f) shows the results of the same experiment for the time-varying Delta Wing dataset, which is described below in Section 6. It is quite challenging from a numerical point of view, as integration may be unreliable in close vicinity of the wing and delivers noisy results. Therefore, this dataset is ideally suited to determine the robustness of our method. The first interesting result is that using the inter-segment angle criterion, we are unable to obtain a correct and accurate surface with a reasonable number of integral curves. If the threshold on γ is chosen very conservative, refinement is very aggressive and available memory is quickly exceeded before the surface computation finishes. Choosing a looser bound, however, does not result in an accurate surface, as the timelines diverge quickly from the reference timelines. Hence, we conclude that the inter-segment angle is not a viable criterion in this setting. For the area and distance criteria, however, the conclusions of the first experiment are essentially confirmed.

5.3 Performance

In general, it is hard to give a direct performance metric for integration-based techniques, since the empiric nature of vector field datasets does not allow for a straightforward estimate of e.g. the length of integral curves. Hence, results for different datasets are not comparable. We therefore limit ourselves to general observations on the performance characteristics of the present scheme.

Our algorithm allows the treatment of even very large time-varying datasets on a typical commodity workstation. The limiting factor in our case is not overall dataset size, but rather the number of time steps that must be kept in memory at any given time to interpolate in the interval $[t^n, t^{n+1}]$ in order to perform numerical integration. Typically, linear or cubic temporal interpolation is used, and hence this number is small. As the time line at t^{n+1} is completed, no longer required time steps can be evicted from main memory and newly required ones can be loaded. Overall, our algorithm possesses streaming characteristics and requires only a single pass over the time steps required to perform the integral surface approximation.

Furthermore, in the case of unstructured meshes, computational cost is dominated significantly by the cost of vector field interpolation. Various approaches exist to accelerate such vector field queries, and we employ a technique by Langbein et al. [9] that makes use of a *kd*-tree based domain decomposition and offers very good performance and reduced memory footprint compared to other such schemes. In addition, by using the DOPRI5 integration scheme, we make maximal use of adaptive integration and avoid unnecessary step reductions that would in turn generate more interpolation queries. The memory overhead introduced by storing complete polynomial representations of individual integral curves until the surface is triangulated is in our experience negligible.

5.4 Datasets

In this section, we briefly describe the datasets we have used for numerical experiments and to generate the images in the remainder of this work.

Ellipsoid The Ellipsoid dataset results from an unsteady simulation of the flow around an ellipsoid, where the angle of the surrounding flow changes over time. Vortex shedding can be observed on the ellipsoid boundary. The data consists of a uniform unstructured mesh of 2.6 million points, over which the flow field is given in 400 time steps.

Delta Wing In order to study the effects of vortex breakdown in aviation, an unsteady simulation of a delta wing configuration exhibiting vortex breakdown was performed. We have selected this dataset since it has proven difficult from a numerical perpective in previous work. Hence, it represents an ideal numerical testcase for the robustness of our method. The dataset consists of 1000 time steps over a constant grid with 18 million tetrahedral elements.

Car This steady simulation models the flow around a car. Vortex shedding can be observed on various parts of the car, and the flow around the rear view mirrors poses a difficult case for adaptive timeline refinement, which our algorithm accomplished well (see Figure 7(b)). The flow vector field is represented on an unstructured mesh with 15 million elements.

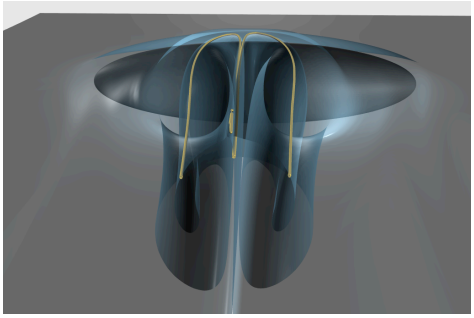


Fig. 5. Path surface self-intersection in the Ellipsoid dataset. Transparent surface ambiguity is resolved by explicit representation of intersection curves (yellow).

6 Visualization

In the following section, we briefly turn our attention to integral surface based visualization and discuss some approaches, and proceed to give some examples.

Transparency and Color Maps Immediate display of integral surfaces allows one to grasp the folding, shearing and twisting nature of the flow sheet spanned by the surface. Such behavior can be observed even in static images through surface depth cues given by lighting and shading (cf. Figure 6(a)). However, typical application vector fields such as those resulting from flow simulations often induce the surface to wrap around itself, creating occlusion problems. Transparent rendering can provide a partial solution in this case. We have however found it beneficial to make use of the natural (s,t)-parameterization of an integral surface to employ gradual color mapping, such that different layers in a transparent image differ by color. This greatly enhances perception of the depth structure of such a transparently rendered integral surface. Figure 6(b) shows the same path surface as Figure 6(a) in a color mapped rendering. The depth of the different layers of the surface wrapping around the forming vortex can be clearly identified by color. Figures 7 demonstrates the same type of rendering on the car dataset.

Self-Intersection In the case of a stationary vector field, a stream surfaces cannot transversally intersect itself or any other stream surface. This is a consequence of the fact that at any intersection point, both surfaces or surface parts must be parallel to the vector field, ruling out the transversality. However, for time-dependent vector fields, this restriction does not hold. We have found that the resulting self-intersections can be visually confusing, especially in transparent surface renderings. We therefore choose to explicitly incorporate the self-intersection curves into the rendering in these cases to resolve the ambiguity created by surface transparency. Figure 5 shows an example from an early time step of the Ellipsoid dataset in which the vortex behind the ellipsoid is in the process of forming.

Texture Mapping Texture mapping provides a simple way to make parts of an integral surface partially or fully transparent. We make use of this property in the following way. By applying a texture map that is fully transparent except for a set of thin lines, and mapping this texture onto the surface such that the lines coincide with surface lines of constant t-parameter, we effectively obtain a depiction of time lines. Conversely, mapping along lines of constant s-parameter results in a depiction of pathline or stream ribbon character. Furthermore, if the texture is continuously shifted along s or t, respectively, an animation is obtained that illustrates the change of such lines as they traverse the surface. Figures 6(c) and 6(d) show the resulting visualizations.

7 Conclusion

In the present work, we have introduced a novel algorithm for the computation of integral surfaces. Our method is based on the clean separation of integral surface approximation from the generation of a graphical representation. This separation allows us to overcome the limitations of earlier techniques and successfully treat the computation of path surfaces in large time-varying datasets. The presented technique is generic with respect to the refinement criteria, and we have examined several choices and provided numerical experiments that allowed us to judge their respective performance as well as verify the overall correctness of our approach.

For future work, we are interested in applying our algorithm to the problem of surface separation and a computation of the resulting separation surfaces in the case of unsteady flows. Regarding the generated surface triangulation, we would like to examine the suitability of post-processing to optimize the triangulation. Furthermore, extension of this work to streak surfaces and time surfaces seems achievable.

Acknowledgements

The authors wish to thank Markus Rütten from DLR Göttingen for supplying some of the datasets treated here and insightful discussion. We are also very much indebted to our colleagues at the Insitute for Data Analysis and Visualization for discussion and feedback.

References

[1] A. A. Andronov. *Qualitative Theory of Second-Order Dynamic Systems*. John Wiley & Sons, 1973.

[2] W. Cai and P.-A. Heng. Principal stream surfaces. In *VIS '97: Proceedings of the 8th conference on Visualization '97*, pages 75–ff., Los Alamitos, CA, USA, 1997. IEEE Computer Society Press.

[3] U. Dallmann. Topological Structures of Three-Dimensional Flow Separations. Technical Report 221-82 A 07, Deutsche Forschungs- und Versuchsanstalt fuer Luft- und Raumfahrt, 1983.

[4] C. Garth, X. Tricoche, T. Salzbrunn, and G. Scheuermann. Surface techniques for vortex visualization. In *Proceedings Eurographics - IEEE TCVG Symposium on Visualization*, May 2004.

[5] I. Gladwell, L. F. Shampine, L. S. Baca, and R. W. Brankin. Practical aspects of interpolation in runge-kutta codes. *SIAM J. Sci. Stat. Comput.*, 8(3):322–341, 1987.

[6] J. Guckenheimer and P. Holmes. *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*. Springer-Verlag, 1983.

[7] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I, second edition*, volume 8 of *Springer Series in Comput. Mathematics*. Springer-Verlag, 1993.

[8] J. P. M. Hultquist. Constructing stream surfaces in steady 3d vector fields. In A. E. Kaufman and G. M. Nielson, editors, *Proceedings of IEEE Visualization 1992*, pages 171 – 178, Boston, MA, 1992.

[9] M. Langbein, G. Scheuermann, and X. Tricoche. An efficient point location method for visualization in large unstructured grids. In *Proceedings of Vision, Modeling, Visualization*, 2003.

[10] R. S. Laramee, C. Garth, J. Schneider, and H. Hauser. Texture advection on stream surfaces: A novel hybrid visualization applied to cfd simulation results. In *Data Visualization, Proceedings of the Joint EUROGRAPHICS - IEEE VGTC Symposium on Visualization (EuroVis 2006)*, 2006.

[11] H. Löffelmann, L. Mroz, E. Gröller, and W. Purgathofer. Stream arrows: enhancing the use of stream surfaces for the visualization of dynamical systems. *The Visual Computer*, 13(8):359 – 369, 1997.

[12] G. M. Nielson. Dual marching cubes. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 489–496, Washington, DC, USA, 2004. IEEE Computer Society.

[13] P. J. Prince and J. R. Dormand. High order embedded runge-kutta formulae. *Journal of Computational and Applied Mathematics*, 7(1), 1981.

[14] T. Schafhitzel, E. Tejada, D. Weiskopf, and T. Ertl. Point-based stream surfaces and path surfaces. In *GI '07: Proceedings of Graphics Interface 2007*, pages 289–296, New York, NY, USA, 2007. ACM.

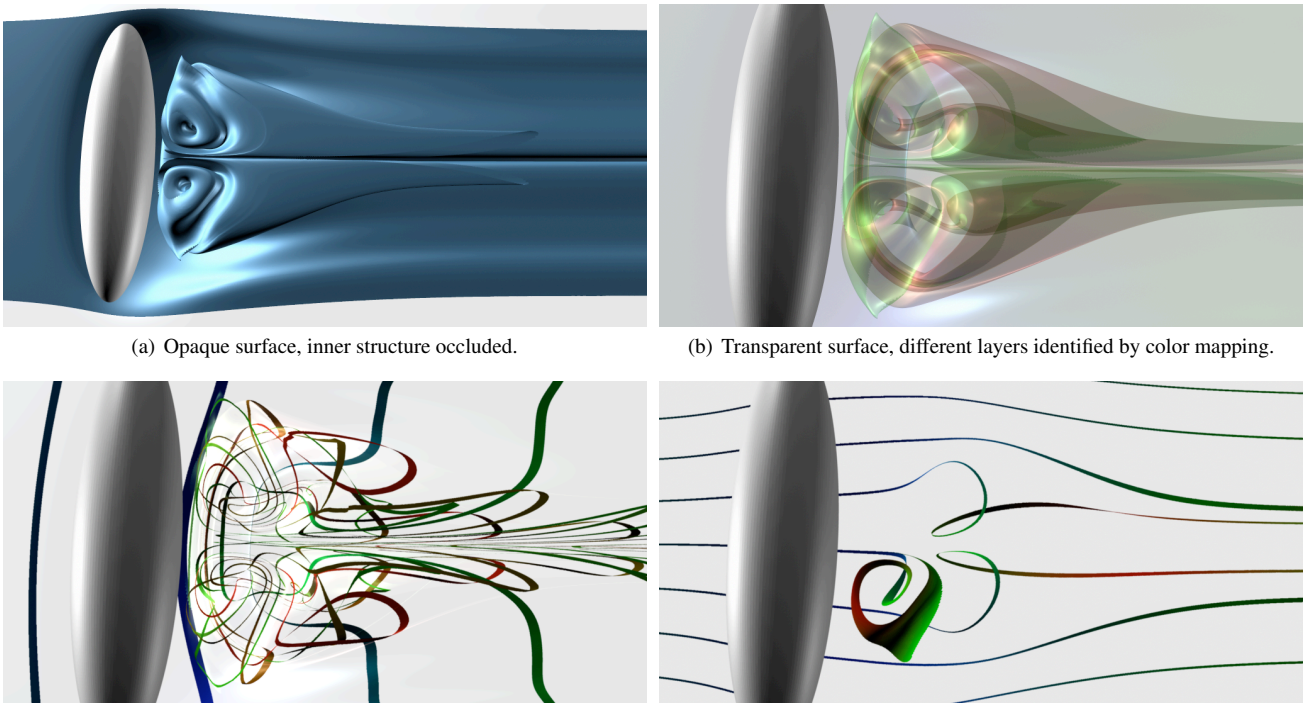
[15] G. Scheuermann, T. Bobach, H. Hagen, K. Mahrous, N. Hahman, and K. Joy. A tetrahedra-based stream surface algorithm. In *IEEE Visualization Proceedings*, 2001.

[16] L. F. Shampine. Interpolation for runge-kutta methods. *SIAM J. Numer. Anal.*, 5, 1985.

[17] D. Stalling. *Fast Texture-Based Algorithms for Vector Field Visualization*. PhD thesis, Freue Universität Berlin, 1998.

[18] J. van Wijk. Implicit stream surfaces. In *Proceedings of IEEE Visualization '93 Conference*, pages 245–252, 1993.

[19] J. J. van Wijk. Rendering surface particles. In *IEEE Visualization Proceedings*, pages 54 – 61, 1992.



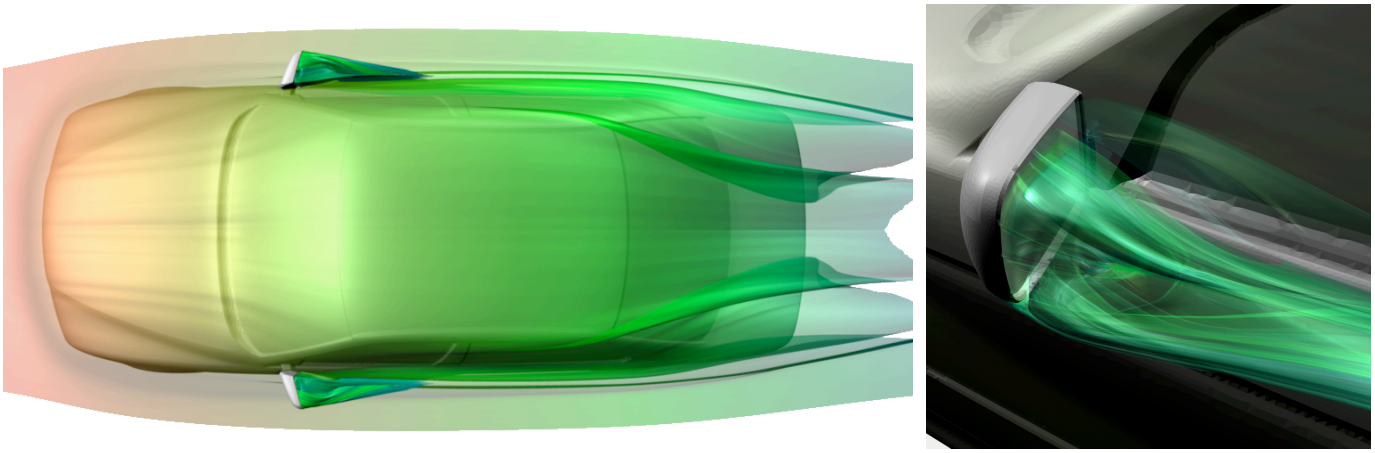
(a) Opaque surface, inner structure occluded.

(b) Transparent surface, different layers identified by color mapping.

(c) Time line texture mapping.

(d) Pathline-type texture mapping.

Fig. 6. Path surface visualization of vortex shedding on an ellipsoid. The surface consists of 508.169 triangles.



(a) Overview with a single stream surface.

(b) Close-up of the flow structure behind the mirror.

Fig. 7. Visualization of flow around a car using a single transparent stream surface, consisting of 1.031.111 triangles.