Robust Polylines Tracing for N-Symmetry Direction Field on Triangulated Surfaces

NICOLAS RAY
INRIA Lorraine
and
DMITRY SOKOLOV
Universite de Lorraine

We are proposing an algorithm for tracing polylines that are oriented by a direction field defined on a triangle mesh. The challenge is to ensure that two such polylines cannot cross or merge. This property is fundamental for mesh segmentation and is impossible to enforce with existing algorithms.

The core of our contribution is to determine how polylines cross each triangle. Our solution is inspired by EdgeMaps where each triangle boundary is decomposed into inflow and outflow intervals such that each inflow interval is mapped onto an outflow interval. To cross a triangle, we find the inflow interval that contains the entry point, and link it to the corresponding outflow interval, with the same barycentric coordinate. To ensure that polylines cannot merge or cross, we introduce a new direction field representation, we resolve the inflow/outflow interval pairing with a guaranteed combinatorial algorithm, and propagate the barycentric positions with arbitrary precision number representation. Using these techniques, two streamlines crossing the same triangle cannot merge or cross, but only locally overlap when all streamline extremities are located on the same edge.

Cross-free and merge-free polylines can be traced on the mesh by iteratively crossing triangles. Vector field singularities and polyline/vertex crossing are characterized and consistently handled.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Curve, surface, solid, and object representations*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Physically based modeling*

General Terms: Algorithms

Additional Key Words and Phrases: Computer graphics, geometry, streamline, N-ROSY, direction field, tangent vector field

ACM Reference Format:

Nicolas Ray and Dmitry Sokolov. 2014. Robust polylines tracing for N-symmetry direction field on triangulated surfaces. ACM Trans. Graph. 33, 3, Article 30 (May 2014), 11 pages.
DOI: http://dx.doi.org/10.1145/2602145

This work was supported in part by INRIA Team ALICE.

Authors' addresses: N. Ray (corresponding author), INRIA Lorraine, France; email: nicolas.ray@inria.fr; D. Sokolov, Universite de Lorraine, France.

DOI: http://dx.doi.org/10.1145/2602145

1. INTRODUCTION

Segmentations of triangulated surfaces that align chart boundaries with a direction field often exhibit useful properties for computer graphics applications. For instance, alignment with the main curvature directions is used for quad-dominant remeshing [Alliez et al. 2003], following the gradient of a scalar field allows to compute pure quad decomposition using Morse-Smale complexes [Dong et al. 2006; Szymczak and Zhang 2012], and streamlines of a cross field can decompose a mesh into quad-shaped domains [Kowalski et al. 2013]. However, it is required that cutting polylines do not merge to obtain a pure quad decomposition. This property is difficult to enforce in the presence of highly perturbed geometry, field singularities (Figure 1), or limit cycles (Figure 11).

All previous algorithms tend to generate polyline crossing or merging. Indeed, when tracing a polyline that converges to a limit cycle, the distance between the polyline and the limit cycle decreases at each loop, until the (floating point) number representation accuracy is reached and the polylines either merge or cross.

A more common source of merges comes from vector fields representations that are polynomial inside each triangle, leading to incompatible directions between pairs of adjacent triangles. As illustrated in Figure 2, and observed in practice in Figure 12, these incompatibilities lead streamlines to converge to an edge. Zhang et al. [2006, Section 6.2] analyse this issue and provide an alternative vector field representation based on local flattening. Our representation differs from theirs, but it also prevents merges or splits along streamlines.

Algorithm Overview. Given a triangulated surface and a direction field in our representation (detailed in Section 2), our method traces cross-free and merge-free polylines that are oriented by the direction field.

We start from a polyline extremity located at barycentric coordinates c (and $1-c$) on halfedge e. The algorithm crosses the triangle associated to e by finding the output point (e', c'), and continues on the next triangle. It stops when the streamline reaches the surface boundary, a sink of the field, or when the polyline's number of segments reaches a user-given limit.

The main difficulty is to determine how each triangle is traversed by the polyline. Our approach (Figure 3) is inspired by EdgeMaps [Bhatia et al. 2011]: we decompose the triangle into pairs of inflow/outflow interval, and define the triangle crossing function by a linear mapping between each inflow interval and its corresponding outflow interval.

The original EdgeMaps algorithm is not guaranteed to produce cross-free and merge-free streamlines because its input is a linear vector field per triangle (subject to field discontinuities along edges), pairing of inflow/outflow interval requires to trace streamlines inside triangles (subject to numerical integration errors), and

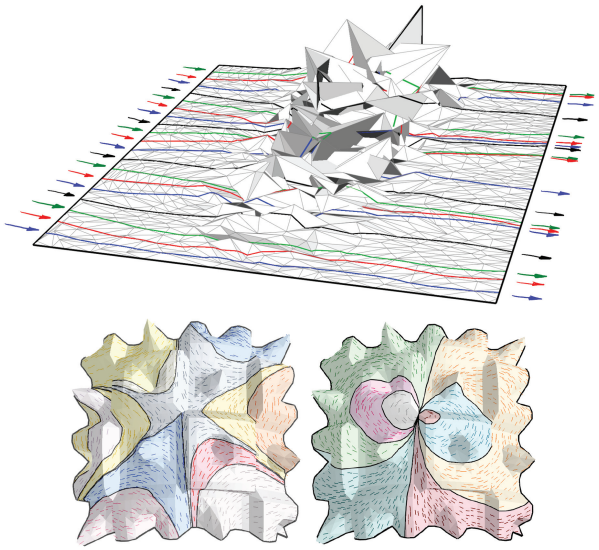
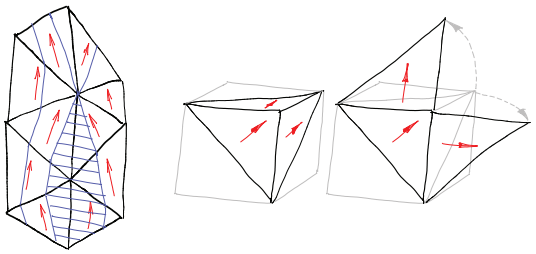


Fig. 1. Our algorithm traces polylines on triangulated surfaces. Unlike previous algorithms, our technique ensures that two polylines cannot cross each other. It works even with highly perturbed surfaces (top) and supports any type of vector field singularities (bottom).



(a) (b)

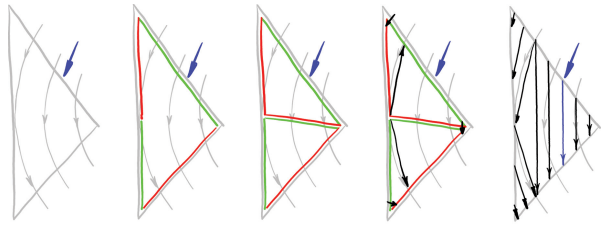
Fig. 2. (a) All streamlines from the dashed area merge on an edge, and split on a vertex; (b) constant per-triangle tangent vector fields have direction discontinuities along edges due to vertex angle defects.

the linear maps between intervals are monotonic only up to numerical precision. We address these issues as follows.

—*Input representation.* We introduce an explicit representation of the field direction on edges, making the field compatible on adjacent triangles.

—*Pairing input/output interval.* The behavior of the field inside each triangle is described by a new structure called stream mesh. The boundary of each (simple) stream face can be decomposed into two regions: one where the field points inside the face and one where field points outside the face (Figure 3(c) and Section 3). The problem of crossing a triangle can then be restated as crossing its stream mesh (Section 4), by iteratively crossing its stream faces. It guarantees the mapping to be monotonic if it is performed with arbitrary precision numbers.

—*Numerical precision.* We use arbitrary precision floating points to represent the barycentric coordinate c, and manipulate them by almost linear mapping functions (Section 5.1) that are guaranteed monotonic.



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

Fig. 3. (a) A polyline (blue arrow) enters a triangle; (b) we construct a stream mesh by a segmentation of the triangle boundary into inflow (green) and outflow (red) segments (Section 3.1); (c) the stream mesh is split into simple stream faces (Section 3.2); (d) we cross the triangle from stream halfedges extremities (Section 4); (e) we map inflow intervals onto outflow intervals using an almost linear mapping with exact precision number representation (Section 5.1).

Practically, the first point prevents merges of the "real streamlines" of the input. The second point allows all nontrivial configurations (field singularities, edge tangent to the field, high-vertices angle defect) to be uniformly and correctly handled without any parameters inherent to numerical integration. The last point allows two polylines to become arbitrarily close to each other, as happens with limit cycles.

Previous Work

To the best of our knowledge, no prior work directly addresses our problem. However, it is interesting to review solutions developed for 2D streamline tracing, to notice similar issues occurring for tracing other types of curves on surfaces, and to give an overview of the tangent vector field and, more generally, N-symmetry direction field design algorithms.

Streamline tracing. Tracing streamlines of 2D or 3D vector fields is a common task [Spencer et al. 2009; Rossl and Theisel 2012] in visualization. In most cases, an order-four Runge Kutta (RK4) integration scheme performs well. For piecewise linear vector field on a triangulation, Bhatia et al. propose EdgeMaps [Bhatia et al. 2011], a more robust solution that directly matches in/outflow intervals of the triangle border. Our method shares the idea of directly mapping in/outflow intervals, but is not limited by vertices angle defect or numerical precision issues.

Tracing curves on triangulated surfaces. Tracing curves on triangulated surfaces is a challenging task because the curve may cross triangles, follow edges, and pass through vertices [Li et al. 2005]. All such configurations are naturally managed by our representation: a polyline passing through a vertex is considered as crossing a subset of its adjacent triangles, all polyline vertices being located on the vertex of the surface.

For computing optimal systems of loops [Colin de Verdière and Lazarus 2005], one needs to distinguish the order between curves following the same edge, leading to a complex data structure where all the curves following the same edge need to be ordered. Special efforts [Martínez et al. 2005; Surazhsky et al. 2005; Polthier and Schmies 2006] have also been devoted to tracing geodesics where the angle defect plays an important role, as in our case.

Recent works [Szymczak and Zhang 2012] compute Morse decomposition of piecewise constant vector fields by converting them

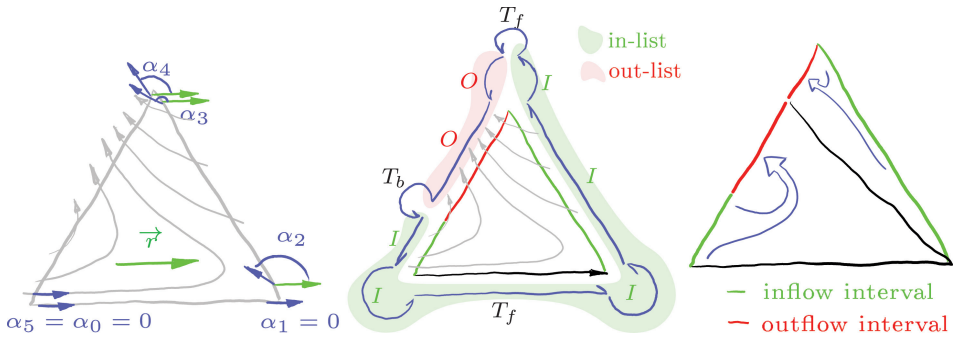


Fig. 4. Left: The field is defined on edge extremities by angles α_i between the field direction and a reference vector $\vec{r}$. Middle: The field behavior with respect to the triangle boundary is explicitly represented by stream halfedges (blue) where the field is either incoming, outgoing, or tangent (I, O, T_f or T_b). In this example, the stream-mesh is a simple stream face: the field enters the triangle from a single triangle section (in-list) and leaves it from a single triangle section (out-list). Right: The black "streamline" is traced from the tangent of the in-list (lower edge). It defines two inflow/outflow pairs of intervals that define the final mappings to be performed with arbitrary precision floating point.

into a combinatorial structure. It results in a robust algorithm, but the streamlines traced from saddles to create edges of the Morse complex still merge of split due to the input field.

Direction field design. Many algorithms [Zhang et al. 2006; Wang et al. 2006; Fisher et al. 2007] allow for the design of tangent vector fields. The resulting field can be continuous enough to have (continuous) streamlines that do not cross one another [Zhang et al. 2006], eventually at the expense of simultaneously refining the surface [Wang et al. 2006]. Albeit, there exists no solution to trace merge-free streamlines.

For mesh segmentation, it is more common to use N-symmetry direction fields than tangent vector fields, but an N-symmetry direction field is equivalent to a vector field on an N-covering of the surface [Kälberer et al. 2007]. Such fields were used for quad remeshing based on global parameterization [Ray et al. 2006]. The lack of control over the topology of these direction fields was addressed later [Palacios and Zhang 2007; Ray et al. 2008]. A common representation [Kälberer et al. 2007; Ray et al. 2008, 2009; Bommes et al. 2009] samples the direction on triangle and makes explicit the field rotation between adjacent triangles.

2. FIELD REPRESENTATION

The continuity of tangent vector fields is naturally defined on smooth surfaces. It is possible to extend this notion of continuity on triangulated surfaces [Zhang et al. 2006, Section 6] by considering that, across an edge, the vector field should preserve its magnitude and the angle with respect to the edge. This is unfortunately impossible to achieve with most existing vector field representations, and results in possible streamline merges. We see this issue in detail and introduce an alternative representation.

Most representations of tangent vector fields are polynomial on each triangle. These vector fields are differentiable everywhere on each triangle, so their direction expressed as an angle in a local basis of the triangle is also differentiable. This continuity of the field on triangles also involves discontinuities of the field direction on edges in the vicinity of vertices with nonzero angle defect. Indeed, along an infinitesimal circle around the vertex, a unit regular vector field will undergo a rotation that is equal to the vertex angle defect. As the field is differentiable on triangles, the direction rotation accumulated along the cycle necessarily comes from direction discontinuities when crossing edges (Figure 2(b)). Such discontinuities can lead to merging streamlines on an edge where the flow leaves both adjacent triangles (Figure 2(a)).

These issues were already addressed in Zhang et al. [2006, Section 6.2], where the field is defined on each vertex by a 2D vector in a local map of its one-ring neighborhood, and interpolated on each triangle. However, numerical approximations of this field's streamlines are not guaranteed not to cross each other.

To avoid the numerical integration of streamlines inside triangles, we rely only on the field direction along edges. Moreover, we prefer to interpolate the field in polar coordinates instead of Cartesian coordinates to allow for more general types of direction field and singularities. This also simplifies the field representation by restricting singularities to be located on vertices.

We represent the input field on each triangle by sampling the field direction at each edge extremity: $\alpha_k, k \in [0 \dots 5]$ are the angles of the field, with respect to a reference vector $\vec{r}$ taken in the triangle plane (Figure 4, left). Note that due to angle defect and singularities on vertex, each triangle corner is associated to two angles: one for each incident edge.

To prevent crossings, we force the input field to be continuous across edges, that is, to have the same angle with respect to an edge on both adjacent triangles of this edge. We also constrain the angle discontinuity on triangle corners to be evenly distributed around each corner. This latter constraint is equivalent to the local flattening of one-ring neighborhoods in Zhang et al. [2006] and allows to better manage singularities (Appendix A.3).

Connection with direction field. We have defined how to represent a vector field on each triangle. To handle N-symmetries, other directions are generated by applying a rotation of $2k\pi/N$ with $k \in 1..N-1$ to the vector field. The vector field across an edge requires the k's of each triangle to agree: their difference (referred to as layer shift in N-coverings) is uniquely defined due to the continuity (enforced in the previous paragraph) of the field across edges.

3. STREAM MESH

A stream mesh is the combinatorial representation of the field behavior inside a triangle of the mesh. It is a halfedge data structure endowed with additional information that represents the field behavior with respect to the triangle boundary. The field direction is given at each stream vertex by its angle α relative to the triangle

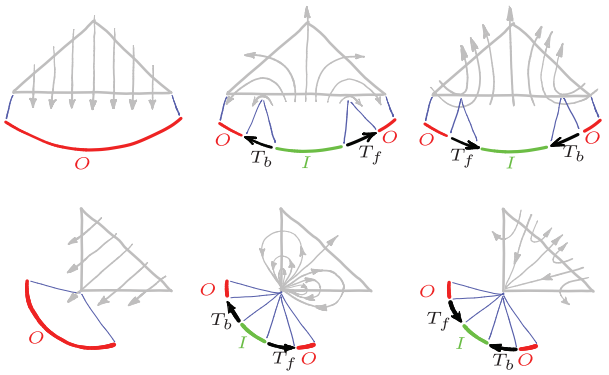


Fig. 5. Combinatorial representation of the flow behavior. The first row shows the decomposition of an edge into incoming (green), outgoing (red), and tangent forward and backward (black arrows) stream halfedges, for three different fields. The second row shows that similar situations can occur on a triangle corner and can be characterized the same way. The field behavior is the same on both rows, but in the second row, the field rotation is performed on a single point instead of a triangle edge. The only difference between columns 2 and 3 is the tangent direction.

reference vector $\overrightarrow{r}$. Along each stream halfedge e, the field has a unique behavior that may be:

—incoming (I) if the field points inwards the stream face,
—outgoing (O) if the field points outwards the stream face,
—tangent in the forward direction (T_f) if the field has the stream halfedge direction,
—or tangent in the backward direction (T_b) if the field direction is opposite to the stream halfedge direction.

As illustrated in Figure 4, middle, stream halfedges represent the behavior of the field with respect to the triangle boundary. They can correspond to a portion of an edge, or be limited to a single point where the field becomes tangent to an edge, or define the field behavior on a triangle corner. In the latter case (Figure 5), multiple stream halfedges are used to describe the field behavior on a single point (triangle corner).

In this representation, we can define:

—an *in-list* as a list of stream halfedges that contains at least one incoming stream halfedge, and no outgoing stream halfedge;
—an *out-list* as a list of stream halfedges that contains at least one outgoing stream halfedge, and no incoming stream halfedge; and
—a *simple stream face* as a stream face having a border that can be decomposed into an in-list, followed by a forward tangent stream halfedge, followed by an out-list, and followed by a backward tangent stream halfedge (Figure 6, right).

The stream mesh is initialized as a single stream face by decomposing the triangle border according to the field behavior (Section 3.1). The main stream face is then decomposed into simple stream faces by a strategy inspired from the ear clipping algorithm [Eberly 1998]: simple stream faces are iteratively removed from the main stream face until the main stream face becomes simple (Section 3.2).

3.1 Main Stream Face Initialization

The initialization of the main stream face of a triangle is performed independently between each pair of field samples. Each such pair corresponds either to a triangle edge or to a corner of the triangle between an edge and the next edge around the triangle.

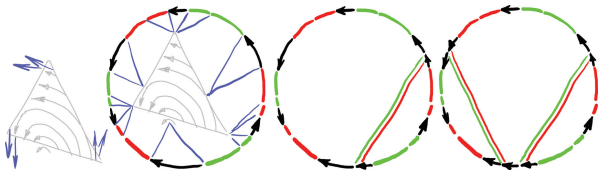


Fig. 6. The field is converted into a stream mesh, then a simple stream face is removed at each step until the main stream face becomes simple.

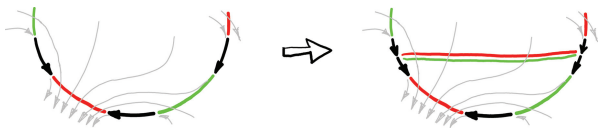


Fig. 7. Splitting the main stream face (left) by our rule produces a simple stream face and removes a pair of in-list/out-list of the main stream face border.

For the k^{th} edge E_k of the triangle, the angle of the field with respect to the edge is given by a linear interpolation between $\alpha_{2k} - \angle(\overrightarrow{r}, \overrightarrow{E_k})$ and $\alpha_{2k+1} - \angle(\overrightarrow{r}, \overrightarrow{E_k})$.

—When this angle equals $0 \mod 2\pi$ it is a forward tangent.
—When it equals $\pi \mod 2\pi$ it is a backward tangent.
—When it is strictly between 0 and $\pi \mod 2\pi$, it is incoming.
—And it is outgoing otherwise.

A stream halfedge is generated for every interval with constant type of behavior, including zero-length intervals when the field is tangent at a single point. These tangent directions must be explicitly represented as illustrated in the first row of Figure 5, where columns 2 and 3 differ only by their opposite tangent directions.

On the triangle corner between k^{th} edge E_k and j^{th} edge E_j (with $j - k = 1 \mod 3$), α_{2k+1} and α_{2j} may be different due to vertex angle defect or field singularities. Consequently, it is possible for a vertex to contain important topologic information about the field. As illustrated in Figure 5, the field behavior on a vertex (second row) is similar to its behavior along an edge (first row), and can be characterized in the same way. The segmentation is performed with the algorithm described for edges, except that angles are linearly interpolated between $\alpha_{2k+1} - \angle(\overrightarrow{r}, \overrightarrow{E_k})$ and $\alpha_{2j} - (\angle(\overrightarrow{r}, \overrightarrow{E_k}) + \angle(\overrightarrow{E_k}, \overrightarrow{E_j}))$. One can notice that using $\angle(\overrightarrow{r}, \overrightarrow{E_k}) + \angle(\overrightarrow{E_k}, \overrightarrow{E_j})$ instead of $\angle(\overrightarrow{r}, \overrightarrow{E_j})$ allows to consider that the triangle border rotation on the corner is in $]0, \pi[$ (no modulo 2π).

A possible geometric interpretation of stream halfedges generated on triangle corners could be to consider the triangle as a rounded triangle having its corner radius tending to 0. It makes the field and the triangle border rotate along the arc of the circle instead of a single point.

3.2 Split the Stream Mesh into Simple Stream Faces

The stream mesh is now initialized by a main stream face. The decomposition iteratively removes simple stream faces from the main stream face until the main stream face becomes simple (Figure 6).

To remove a simple stream face (Figure 7), we search in the stream halfedges list of the main stream face a sequence of halfedges that can be decomposed into a T_f stream halfedge, followed by an

out-list, followed by a T_b stream halfedge, followed by an in-list, and followed by a T_b stream halfedge. We split the first T_f and last T_b stream halfedges of the sequence and introduce a new stream edge linking the stream vertices produced by the stream edge split. The type of the generated stream halfedges is set to *incoming* in the simple stream face side, and *outgoing* in the main stream face side.

As illustrated in Figure 7, the type of the produced stream halfedges is coherent with the flux that can be computed across the stream halfedge. Indeed, the triangle border being convex, the field direction at the new stream halfedge's extremities will always point to the same halfplane of the new stream halfedge.

By symmetry, it is also possible to apply the same operation on the opposite field, that is, replace both $T_f \Leftrightarrow T_b$ and in-list$\Leftrightarrow$ out-list in the pattern and in the result.

Recursively applying the split operation converges to a decomposition into simple stream faces, as demonstrated in Appendix B.1.

4. PAIRING INTERVALS

Pairing inflow/outflow intervals using the original EdgeMaps algorithm [Bhatia et al. 2011] requires to trace a set of "streamlines" inside each triangle. However, the numerical imprecision involved by the streamline integration inside triangles may produce an invalid decomposition. Typical failure cases include high field rotation close to field singularities, or fields that are almost tangent to an edge. To prevent such failure cases, we replace the numerical streamline integration by a traversal of the stream mesh.

A robust method crosses each simple stream face by almost linear mappings (Section 5.1) between their in-list and out-list. This solution guarantees the generation of intersection-free streamlines, but the polyline orientation may not closely match the direction field geometry.

Alternatively, a more geometric method crosses each simple stream face by using a heuristic to take the field geometry into account. This method may fail due to numerical approximations, but better fits the field geometry.

Both solutions only differs by the estimation of the direction field flux Φ across the stream halfedges. We start with the geometric heuristic and switch to the robust version if needed.

Remark 1. At this point, we have two solutions to cross a triangle, but we don't use them directly for tracing the polyline because the robust version has a poor geometry with respect to the direction field, and the other one may result in crossing streamlines. Instead we use this method only to decompose the triangle boundary into inflow/outflow intervals (Figure 4, right) then use EdgeMaps with arbitrary precision to perform the final mapping.

Remark 2. The geometric heuristic with fixed precision is usually good enough for the decomposition, but not for directly the tracing streamline. On one hand, the decomposition requires to trace only few "streamlines", and it is possible to check the validity independently in each triangle. On the other hand, tracing a polyline is much more difficult because each new segment must be guaranteed not to cross all previous and future segments that may cross this triangle.

4.1 Crossing a Simple Stream Face

Crossing a simple stream face requires to define how points in the in-list are mapped to points in the out-list. Any such mapping that does not cross streamlines will produce globally cross-free streamlines. However, it is better to choose a mapping that preserves as much as possible the field geometry. Our mapping is defined such that any

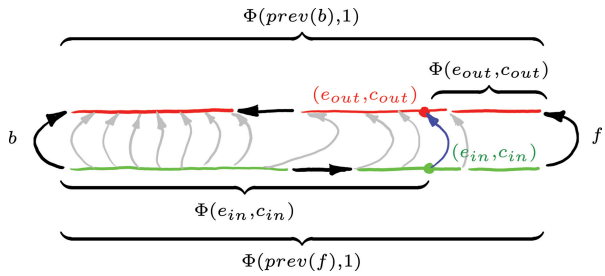


Fig. 8. Flow notations used to cross a simple stream face.

evenly distributed set of streamlines that enters a triangle will leave it with an even distribution, except if field sinks or streamlines that are tangent to the boundary prevent it. It can be restated as follows: for a unit norm field, if the stream face is split by a streamline, both parts should have the same ratio between the incoming flux and the outgoing flux. Here, we call by flux the amount of streamlines outgoing from a portion of the out-list (and symmetrically for the in-list). However, it can be considered as an abuse of terminology because we explicitly set a nonzero flux for sink/source vertices to allow for an infinite set of streamlines to pass through them (Section 4.2), whereas computing the flux of the unit vector field would give zero. This heuristic perfectly respects the field when it is constant inside the triangle, and is evaluated in Section 6.1 in more difficult situations.

As illustrated in Figure 8, we call f (respectively, b) the stream halfedge of type T_f (respectively, T_b) that comes before the out-list (respectively, in-list).

We denote by $\Phi(e, c)$ the flux crossing the in-list (respectively, out) of stream halfedges up to the point located at the $(c, 1-c)$ barycentric coordinate on the stream halfedge e. It is recursively defined by $\Phi(e, c) = \Phi(prev(e), 1) + \phi_e(c)$ where $\Phi(f, 1) = 0, \Phi(b, 1) = 0$, and $\phi_e(c)$ is the flux crossing the stream halfedge e up to the point of barycentric coordinates $c, 1-c$.

Using these notations (Figure 8), the condition for a streamline to split the simple stream face into two stream faces having the same ratio between inflow and outflow writes

$$\frac{\Phi(e_{in}, c_{in})}{\Phi(prev(f), 1)} = 1 - \frac{\Phi(e_{out}, c_{out})}{\Phi(prev(b), 1)},$$

where the input point is e_{in}, c_{in} and the output point is e_{out}, c_{out}. As a consequence, the output point is given by:

$$(e_{out}, c_{out}) = \Phi^{-1}\left(\Phi(prev(b), 1)\left(1 - \frac{\Phi(e_{in}, c_{in})}{\Phi(prev(f), 1)}\right)\right).$$

To compute the output position (e_{out}, c_{out}) of a streamline, we need to evaluate the functions Φ, and Φ^{-1}. The function Φ can be evaluated from $\phi_e(c)$ using its recursive definition. The function $\Phi^{-1}(x)$ requires to take the stream halfedge e such that $\Phi(e, 0) \leq x \leq \Phi(e, 1)$ and $\phi_e(1) \neq 0$, and to define its barycentric coordinate $c = \phi_e^{-1}(x - \Phi(e, 0))$.

As a consequence, we only need to be able to evaluate $\phi_e(c)$ and its inverse $\phi_e^{-1}(x)$ to cross a simple stream face. For the robust version, it is sufficient to set $\phi_e(c) = \phi_e^{-1}(c) = c$, and for the heuristic version it is described in Section 4.2 for $\phi_e(c)$ and Section 4.3 for $\phi_e^{-1}(c)$.

4.2 Computing $\phi_e(c)$

To estimate a flux across edges, the vector field orientation (direction field) is not sufficient, therefore we also assume that its magnitude

is equal to one. On edges, we set $\phi_e(c)$ to be the flux of this vector field across the stream halfedge e given by

$$\begin{aligned}\phi_e(c) &= |\overrightarrow{e}| \int_0^c -\sin(\alpha_o + t(\alpha_d - \alpha_o) - \angle(\overrightarrow{e}, \overrightarrow{r}))dt, \\ &= |\overrightarrow{e}| \frac{\cos(\alpha_o + t(\alpha_d - \alpha_o) - \angle(\overrightarrow{e}, \overrightarrow{r}))}{\alpha_d - \alpha_o}\Big|_0^c,\end{aligned}$$

where α_o and α_d are the field directions located at the vertex pointed by the stream halfedges $prev(e)$ and e, and expressed by their angle relative to $\overrightarrow{r}$.

On corners, we can generally say that there is no flux that leaves the triangle, that is, $\phi_e(c) = 0$. However, for singularities with positive index such as source and sinks, there is an infinity of streamlines that reach or start from the corner (Figure 19). If an outflow stream halfedge e is defined in a triangle corner, in a sequence T_f, O, T_b, then we set $\phi_e(c) = c$. By symmetry, if an inflow stream halfedge e is defined in a triangle corner, in a sequence T_b, I, T_f, then we set $\phi_e = -c$. This strategy provides a field behavior coherent with the continuous behavior of streamlines on field singularities as explained in Appendix A.3.

4.3 Computing $\phi_e^{-1}(x)$

Computing $\phi_e^{-1}(x)$ requires to invert Eq. (1). As cosine is not a one-to-one function, determining $\phi_e^{-1}(x)$ requires to take into account that it is a barycentric coordinate in the halfedge e, and therefore $0 \leq \phi_e^{-1}(x) \leq 1$. This constraint fixes $s \in \{-1, 1\}$ and $k \in \mathbb{Z}$ in the formula:

$$\begin{aligned}\phi_e^{-1}(x) &= \frac{s \arccos(\cos(\alpha_o - \angle(\overrightarrow{e}, \overrightarrow{r}_T)) - x\frac{(\alpha_d - \alpha_o)}{|\overrightarrow{e}|})}{\alpha_d - \alpha_o} \\ &+ \frac{2k\pi - \alpha_o + \angle(\overrightarrow{e}, \overrightarrow{r}_T)}{\alpha_d - \alpha_o}.\end{aligned}$$

5. CROSSING TRIANGLE WITH ARBITRARY PRECISION

When a polyline reaches a triangle edge e at barycentric position p (given in arbitrary precision floating point) on e, we are able (Section 4) to determine the corresponding inflow interval (barycentric coordinates $[a/2^i, b/2^i]$ on e) and outflow interval (barycentric coordinates $[c/2^j, d/2^j]$ on edge e'). The usage of dyadic rationals (denominator is a power of two) is motivated by the direct compatibility with floating points, and possible simplifications exploited in our almost linear mapping.

The objective is to determine the barycentric coordinate q on edge e'. A linear interpolation gives $q = c/2^j + (p - a/2^i)(d/2^j - c/2^j)/(b/2^i - a/2^i)$. However, doing so in exact arithmetic dramatically affects the performances: the memory required to represent q is approximately 100 bits larger (50 for the denominator, and 50 for the nominator) than for p. After crossing n triangles, the size of q is approximately $100n$ bits, which greatly reduces the performance and increases the memory required to store the polyline (Figure 10).

Our solution, described in the next section, is an approximation of a linear mapping that reduces by two orders of magnitude the size of p and q (Figure 9). At coarse scale, it is linear up to approximations of 64-bits floating points, and at finer scale, it is linearly interpolated between the closest 64-bits floating points numbers.

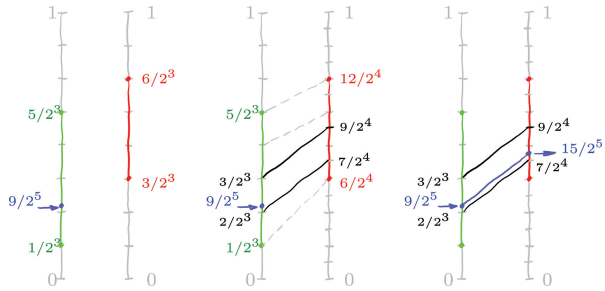


Fig. 9. An example of execution of Algorithm 1. (left) Origin interval $[a/2^i, b/2^i] = [1/2^3, 5/2^3]$, destination interval $[c/2^j, d/2^j] = [3/2^3, 6/2^3]$ and an entry point $p = 9/2^5$. (middle) The destination interval is refined (c and d are doubled and j incremented) and we map points $p' = 2$ and $p'' = 3$ to points $q' = 7$ and $q'' = 9$, respectively. (right) The resulting point $q/2^l = 15/2^5$ is obtained by the linear mapping of the interval $[p'/2^i, p''/2^i]$ onto the interval $[q'/2^j, q''/2^j]$.

5.1 Almost Linear Mapping

In this section we describe how to define a strictly monotonic mapping of all dyadic rationals of an origin interval $[a/2^i, b/2^i]$ to a destination interval $[c/2^j, d/2^j]$, where $a, b, c, d \in \mathbb{N}$ and $a < b, c < d$. Algorithm 1 gives an implementation, and Figure 9 illustrates an example of execution.

Input/output interval boundaries define two grids of fractional numbers with given precision (respectively, 2^i and 2^j). The idea is to refine the output grid until it becomes larger than the input one, and then to map the input grid onto the output grid by rounding the linear mapping (Figure 9, middle). The final mapping is defined as a collection of linear transformations between pairs of grid segments (Figure 9, right).

ALGORITHM 1: Algorithm overview

Input: Origin interval boundaries $[a/2^i, b/2^i]$
Input: Point $p/2^k \in [a/2^i, b/2^i]$ with $k \geq i$
Input: Destination interval boundaries $[c/2^j, d/2^j]$

Output: Point $q/2^l \in [c/2^j, d/2^j]$

```
1  while b − a > d − c do
2  |   (c, d) ← 2 · (c, d);
3  |   j ← j + 1;
4  end
5  while 2 · (b − a) < d − c do
6  |   (a, b, p) ← 2 · (a, b, p);
7  |   (i, k) ← (i + 1, k + 1);
8  end
9  p′ ← ⌊p/2^{k−i}⌋;
10 p″ ← p′ + 1;
11 q′ ← ⌊(p′ − a) · (d − c)/(b − a)⌋ + c;
12 q″ ← ⌊(p″ − a) · (d − c)/(b − a)⌋ + c;
13 q ← (p − p′ · 2^{k−i}) · (q″ − q′)/(p″ − p′) + q′;
14 l ← j + k − i;
15 return q/2^l;
```

The section between lines 5 and 8 of Algorithm 1 is a loop that refines the input grid. While the loop is not mandatory to define

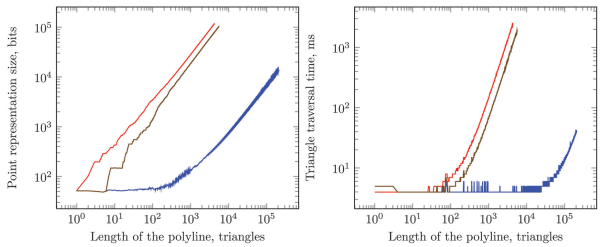


Fig. 10. Results after one hour of computing: (a) length of a polyline (in triangles, x-axis) versus number of bits to represent the current polyline vertex (y-axis): linear mapping (red), without input grid refinement (brown), our algorithm (blue); (b) length of a polyline (in triangles, x-axis) versus time to cross one triangle (y-axis).

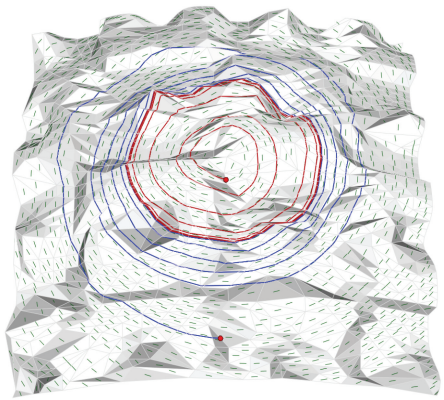


Fig. 11. Robustness stress test: the polyline initialized at the red dot converges to a limit cycle. Our arbitrary precision representation of the polyline prevents crossings and merging whereas 64-bits precision leads to a merge after less than 10 loops. On the same data, we run with exact precision up to 900 loops for generating Figure 10.

a correct mapping, it is essential to save the memory. Indeed, the increase in the precision of the point $q/2^l$ with respect to the point $p/2^k$ is given by the relation $l-k = j-i$, thus we keep i as high as possible to avoid wasting memory. Figure 10 gives a plot of occupied memory for a polyline in the limit cycle field (Figure 11). Note that the growth is not monotonic. This is due to two phenomena: either $i > j$ or a fraction that can be canonicalized.

6. DISCUSSION

This section evaluates the performances of our algorithm on synthetic stress tests (Section 6.1), proposes some applications where tracing robust polylines is required (Section 6.2), compares with possible alternative algorithms (Section 6.3), and provides some details about local overlaps (Section 6.4).

6.1 Synthetic Tests

To evaluate the geometric quality of our polylines, we traced them on a circular vector field with different mesh quality (Figure 13). It shows our polylines, smoothness and accuracy with different triangle qualities (upper to lower) and different field rotation magnitude (border to center). In practice, computer graphic meshes are closer to the upper and middle images, and field design algorithms tend to produce as smooth as possible fields. It is interesting to notice that an important loss of accuracy only appears on very stretched triangles like one having a corner with a field singularity (see close-up).

The cross-free and merge-free properties are ensured by our approach. Figure 1(top) and Figure 11 show examples where these properties are hard to enforce due to noisy geometry and the very short distance between polylines.

6.2 Applications

We illustrate two possible applications of our method: computing quadrangulations inspired by Morse-Smale complexes (Figure 14), and splitting a mesh according to a direction field. Tracing streamlines of an N-symmetry direction field [Kowalski et al. 2013] allows to partition 2D meshes. To illustrate a possible application of our method, we applied the same strategy on 3D surfaces, by growing all streamlines simultaneously, and stopping them when they reach a streamline defined on a perpendicular direction. As a result (Figure 15) we obtain quadrangular charts with T-junctions everywhere except when a degeneracy is prescribed by feature curves as in the fandisk model. Such T-meshes could be useful after optimization, as proposed in Myles et al. [2010].

6.3 Alternative Algorithms

We have proposed the first algorithm that guarantees non-crossing (or merging) streamlines. However, in many applications, alternative algorithms can produce similar results. We review an existing algorithm [Bhatia et al. 2011], a fair solution obtained by combining order-four Runge-Kutta with a continuous vector field representation [Zhang et al. 2006], and our algorithm without adaptive numerical precision. The failure cases are illustrated in Figure 12, and the number of failures on a set of models are given in Figure 16.

—*EdgeMaps.* EdgeMaps requires a linear vector field on each triangle. To produce it, we start from our smooth field, and set the vector on the i^{th} triangle corner to be equal to $(\alpha_{2i} + \alpha_{2i+1})/2$. This strategy is fair as it evenly distributes the angle defect of each vertex over all adjacent triangles (much better than a projection).

On surfaces without angle defect, it offers the same guarantee as our algorithm without adaptive resolution. On other surfaces, streamlines can converge to an edge as in Figure 2, left. In practice, one could expect the failure case to appear very rarely because the streamline must cross an edge with an angle lower than a portion of the angle defect of an incident vertex. However, our experiments illustrated in Figure 16 demonstrate the opposite.

—*RK4 on Zhang et al. [2006].* The field introduced in Zhang et al. [2006] is sufficiently continuous to have noncrossing streamlines. The problem is therefore to determine how often approximations of these streamlines (computed by RK4) do cross each other. We don't exactly use their field representation: we perform the field interpolation on triangles in polar coordinates instead of Cartesian coordinates. This minor modification allows to work with direction fields, constraints singularities to be on vertices, allows to represent fields with high curvature, and is directly compatible with our field, resulting in more fair comparisons.

To trace streamlines on this field representation, we used an order-four Runge-Kutta algorithm. This numerical integration scheme comes with the usual numerical imprecision, the difficulty to tune the time-step parameter, and some thresholds required to manage singular points (reaching a sink, crossing a saddle vicinity, etc.). Moreover, to work on a triangulated surface, the algorithm must deal

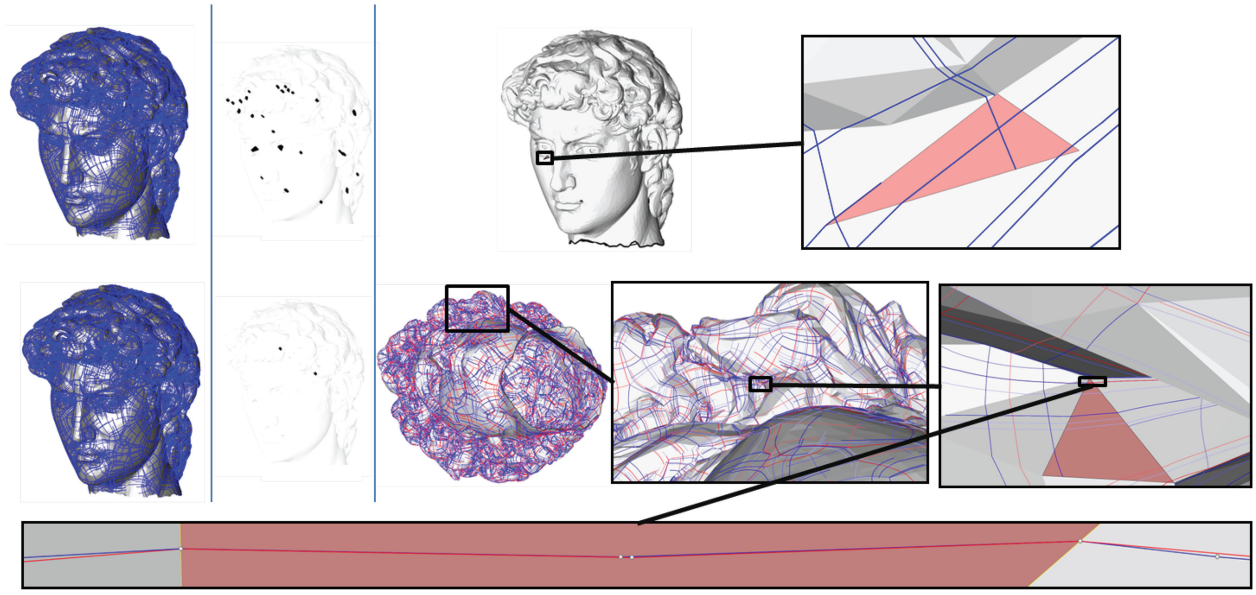


Fig. 12. Crosses and merges of streamlines observed with alternative algorithms. EdgeMaps (top row) create many streamlines (second column) that converge to an edge due to the piecewise linear representation of the vector field. Streamlines traced on a continuous field representation [Zhang et al. 2006] with an order-four Runge-Kutta (second row) lead to only two failures (second column) illustrated by the streamline switch observed in the close-up.

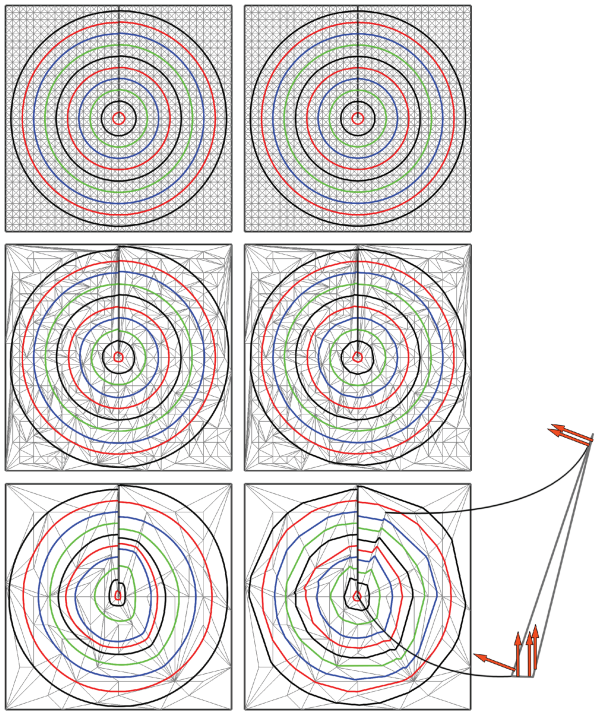


Fig. 13. Our algorithm on the mesh (right column) is compared with a numerical integration (RK4) on the same data (left column), with decreasing mesh quality from top to bottom.

with high curvature due to the angle defect distributed on triangle corners, and numerical ambiguities (e.g., when the streamline has to follow sharp edges of a geometric feature). In our experiments, we use an average of 10 integration steps by triangle, and don't consider

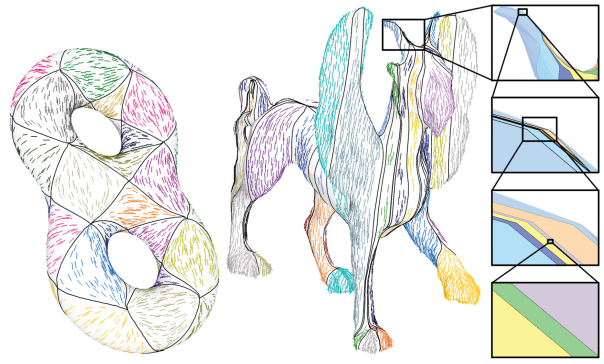


Fig. 14. Morse-Smale complexes provide a quad-shaped charts decomposition of a smooth manifold. Converting a scalar field gradient into our representation allows to have this property for triangulated surfaces. We used a Laplacian eigenfunction for the double torus and the z coordinate for the Feline. Close-ups allow to see that polylines can be very close to each other.

as errors crossings that occur in the one-ring of singular vertices (different strategies would have led to very different results).

In practice, with a smooth vector field, the numerical errors are similar to our algorithm without adaptive resolution ($\approx$10 loops on the spiral model). However, two proximal polylines traced with opposite directions are more likely to cross because RK4 evaluates the field at different positions (see Figure 12).

With this approach, it is possible to decrease the time step to reduce the probability of crossings. However, it doesn't guarantee that no new crossings will appear (Figure 17), and it requires to relaunch the crossing streamlines (and cancel all computations based on them).

—*Our algorithm without adaptive resolution.* This solution is fair as long as we don't reach extreme cases as in the spiral test.

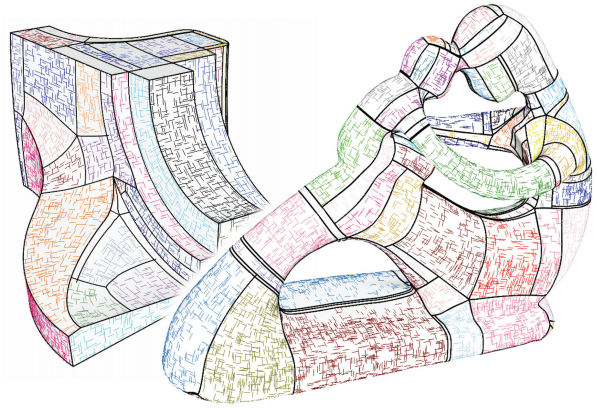


Fig. 15. Tracing streamlines (black curves) from singularities of a cross field provides a decomposition of the surface.

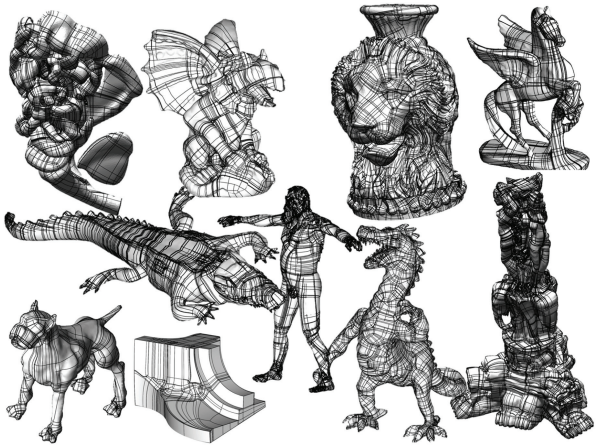


Fig. 16. Number of failures using [EdgeMaps – RK4] : Jobard's plume 12–0, gargoyle 1–0, lionvase 12–0, pegasus 3–0, dog 0–0, crocodile 17–4, fandisk 0–0, monkey man 18–1, dragon 23–0, statue with elephant 66-6.

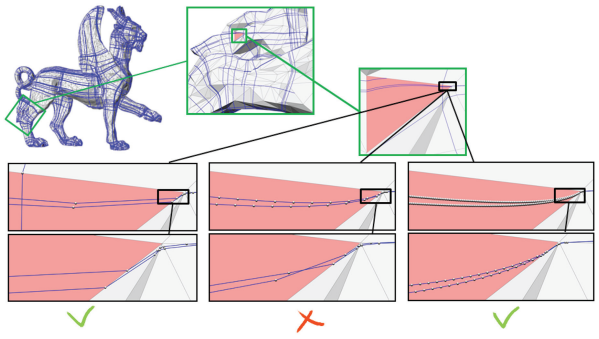


Fig. 17. Impact of RK4 time-step parameter. A triangle is crossed with 10, 100, and 1000 integration steps by triangle (in close-ups). A cross is detected with 100 integration steps.

In this configuration, we obtain a merge after ten loops, which is equivalent to the RK4 solution (tested with 10, 100, or 1000 integration steps by triangle). In practice, it doesn't produce errors on all other tested examples. It also does not have any issues in the vicinity of singular vertices, and does not require any parameter tuning.

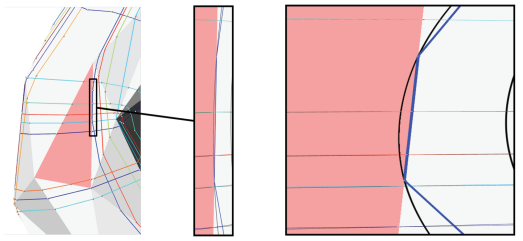


Fig. 18. The blue streamline enters and leaves the pink triangle on the same edge. As illustrated in the stretched version of the close-up (right), the continuous streamline (in black) is approximated by a segment (blue) contained in the edge.

Our algorithm is the only one to ensure that polylines will never cross or merge. However, it could also work without adaptive resolution for common applications. A fair alternative solution would be to extend EdgeMaps to work with Zhang et al. [2006] field representation, and eventually our adaptive number representation. However, this latter solution would rely on RK4 to trace streamlines inside each triangle (to define the edge map), and it would be difficult to prove that no cross/merge could occur here.

6.4 Local Overlaps

When a streamline enters and leaves a triangle on the same edge, the generated segment is localized on the edge (Figure 18). If two such streamlines are traced, they may locally overlap on the triangle edge. However, if polylines are dedicated to cut the mesh into pieces, such overlaps will result in faces with degenerated geometry, but the desired topology. Another side-effect of representing streamlines by a single segment on each crossed triangle is that some points inside the triangle are not covered by polylines, as in the lower part of the triangle in Figure 4, right.

Conclusion. Tracing intersection-free polylines makes it easier to design new algorithms inspired by the continuous settings. Possible improvements of the method include using polycurves inside triangles, or finding a simpler way to cross each triangle. The question of the generalization to higher dimension arises naturally, but it is important to remember that the main issue (angle defect) requires that the metric is not induced by the object itself (for surfaces, it is induced by its embedding in 3D space, but volumes in 3D do not have this issue).

APPENDIXES

A. BEHAVIOR ON VERTICES

A.1 Vertex Indices

N-symmetry direction fields may have singularities that can be characterized by their index. The index is well defined for smooth manifolds [Mrozek 1995], and has been extended to triangulated surfaces [Ray et al. 2009]. In our case, we assume that singularities can only appear on vertices, leading to the following characterization of indices

$$Index(A) = \sum \frac{\Delta\alpha_e}{2\pi} + \frac{2\pi - \sum \beta_e}{2\pi}, \quad (1)$$

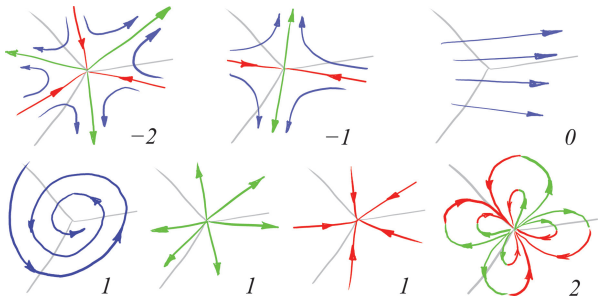


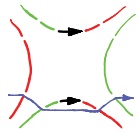
Fig. 19. Singularities classified by index. On negative indices, there exist a finite number of streamlines (red and green) having the vertex as extremity. On regular vertices (index is zero), at most one streamline can cross the vertex. On positive index singularities, there exist an infinity of streamlines having the vertex as extremity, expect for the vortex case (lower left).

where the sums are performed on all triangle corners referred by their halfedge e incident to A, $Index(A)$ is the index of vertex A, $\Delta\alpha_e$ is the angle discontinuity on the triangle corner, β_e is the triangle corner angle. The first sum is the total amount of field rotation around A, and the rest is the angle defect of A divided by 2π.

Examples of singular vertices are given in Figure 19. One can notice that an infinite number of streamlines can reach the vertex only for strictly positive indices, leading to two different behaviors of our algorithm as detailed next.

A.2 Geometric Vertex Crossing

The default behavior of our algorithm is when there is not an infinity of streamlines having the vertex as one of its extremities. In this case, when a streamline leaves a triangle on a vertex location, the output of the triangle crossing algorithm is an adjacent edge, with a barycentric coordinate being either 0 or 1 to fit the vertex location. The streamline then continues on the next triangle until it ends in the vertex or leaves the vertex location as illustrated in the inset figure.



Our algorithm has this behavior because the flux on a stream halfedge defined on a triangle corner is generally zero, and the constraint that the simple stream face crossing algorithm is not allowed to generate outputs on a stream halfedge without flux.

A.3 Streamline Extremity on a Vertex

Streamlines may also have one of their extremities located on a vertex, but this occurs only for vertices with strictly positive index, as illustrated in Figure 19 (we consider that if a unique streamline reaches the vertex it will cross it with the previous behavior). We explain here why our way to determine the flux on stream halfedges inside triangle corners (Section 4.2) gives nonzero flux only for vertices with strictly positive index.

As the rotation speed of the field around the vertex A is constant, the difference of angle $\Delta\alpha_e$ is equal to the sum of such rotations around the vertex A times the ratio of β_e over the sum of triangle corner angles around A. Putting it together with Eq. (1), with summation over all halfedges e' pointing to A gives

$$\Delta\alpha_e = \frac{\beta_e}{\sum \beta_{e'}} \left(2\pi(Index(A) - 1) + \sum \beta_{e'}\right)$$

so the variation of angle with respect to halfedges pointing to A is

$$\Delta\alpha_e - \beta_e = \frac{2\pi\beta_e}{\sum \beta_{e'}}(Index(A) - 1).$$

As a consequence, if $\Delta\alpha_e - \beta_e$ is strictly positive, the vertex index is greater or equal to 1. Otherwise, the index is strictly less than 1. Note that for direction fields with rational indices, we are still able to distinguish between singularities with and without flux.

In our algorithm, the condition to associate some flux to output stream halfedges (defined on a triangle corner) is that the stream halfedge must be contained in a sequence $T_f O T_b$. It means that the field angle with respect to the triangle border increases at least by π. Since the corner is convex, we have $\beta_e < \pi$. As a consequence, our algorithm gives some flux only for stream halfedges in triangle corners corresponding to a vertex with strictly positive index. The same thing occurs for the sequence $T_b I T_f$.

A.4 Starting a Streamline from a Vertex

For a vertex that is the origin of a finite number of streamlines (negative or null index), it is possible to generate all streamlines by simply starting a streamline for each inflow stream halfedge on adjacent triangle corners. This is especially important for tracing streamlines from saddle points, as required for computing Morse-Smale complexes.

B. CORRECTNESS OF THE DECOMPOSITION

B.1 Convergence

Given a stream face with n in-lists and n out-lists, let us choose one out-list as a reference. Any two adjacent lists i and $i+1$ have a tangent between them, let us define a sequence of labels $\{t_i\}_{i=0}^{\infty}$ as the label of tangent stream halfedge incident to both lists i and $i+1$. Then we define a sequence of integers $\{a_i\}_{i=0}^{+\infty}$ as follows.

$$a_0 = 0$$

$$a_{2i+1} = \begin{cases} a_{2i} + 1 & \text{if } t_{2i+1} = T_f, \\ a_{2i} - 1 & \text{otherwise.} \end{cases}$$

$$a_{2i+2} = \begin{cases} a_{2i+1} + 1 & \text{if } t_{2i+2} = T_b, \\ a_{2i+1} - 1 & \text{otherwise.} \end{cases}$$

The defined sequence $\{a_i\}$ is arithmetic quasiperiodic: $a_{i+2n} = a_i - 2$ and is continuous in the sense that $|a_{i+1} - a_i| = 1$. A stream face is simple if and only if the corresponding sequence $\{a_i\}$ is decreasing. The splitting rule described in Section 3.2 searches for a pattern (per period $2n$) $(2i+1, 2i, 2i-1, 2i)$ in the sequence $\{a_i\}$ and replaces it with a new one $(2i+1, 2i)$. In other words, the splitting rule removes one (per period) local minimum of the sequence $\{a_i\}$. The symmetric rule replaces $(2i+2, 2i+1, 2i, 2i+1)$ with $(2i+2, 2i+1)$, again removing a local minimum. If a stream face is not simple, the corresponding sequence has at least one local minima, moreover, the sequence decreases by 2 with each period and therefore it is possible to apply one of the splitting rules. Both rules keep the continuity of the sequence, and the period is reduced by 2 with each iteration, leading to a final decomposition of the initial stream face into a set of simple stream faces.

B.2 Nonnullity of Flux through Simple Faces

We show that each simple stream face is traversed by some flux. To do so we demonstrate that the out-list (as well as in-list) of a simple stream face have nonzero associated flux.

First of all, let us note that all stream halfedges created by splitting rules have nonzero flux. Indeed, their length is not zero: it is easy to see that due to the linear interpolation between angle samples, the sequence $\{a_i\}$ is monotonic inside triangle corners; however, the splitting rule searches for a local minimum of the sequence. Therefore, it is not possible to create a simple face entirely contained in a triangle corner.

Now let us show that all simple faces have nonzero flux through them. Let us suppose that the out-list of a simple stream face has a zero flux. All outflow stream halfedges on triangle edges as well as outflow stream halfedges corresponding to splits have nonzero flux, since their length is greater then zero. The only option for an out-list to have a zero flux is to be contained in a triangle corner and to have T_b, O, T_f structure, as defined in Section 4.2. However, this means that the corresponding sequence $\{a_i\}$ is increasing on this out-list, and that contradicts the monotonicity of the sequence $\{a_i\}$ for simple faces. Therefore, there is no out-list in a simple face that does not have a flux through it. The same argument shows by symmetry that there is no in-list without flux through it.

ACKNOWLEDGMENTS

The 3D models come from the aim@shape project (gargoyle, lionvase, fandisk, pegasus), the Digital Michelangelo Project (David's statue), The Princeton Shape Benchmark (yeti), the Stanford Repository (Thai statue), the Stanford's Computer Graphics Group (feline), free section of commercial Web sites http://artist-3d.com/free_3d_models/dnm/model_disp.php?uid=2106 (dog) and http://artist-3d.com/free_3d_models/dnm/model_disp.php?uid=2106 (dragon), http://www.archibaseplanet.com/download/c45cff28.html (alligator) and Bruno Jobard (plume).

REFERENCES

P. Alliez, D. Cohen-Steiner, O. Devillers, B. Levy, and M. Desbrun. 2003. Anisotropic polygonal remeshing. *ACM Trans. Graph.* 22, 3, 485–493.

H. Bhatia, S. Jadhav, P.-T. Bremer, G. Chen, J. A. Levine, L. G. Nonato, and V. Pascucci. 2011. Edge maps: Representing flow with bounded error. In *Proceedings of the IEEE Pacific Visualization Symposium (PacificVis'11).* 75–82.

D. Bommes, H. Zimmer, and L. Kobbelt. 2009. Mixed-integer quadrangulation. *ACM Trans. Graph.* 28, 3, 77:1–77:10.

E . Colin de Verdiere and F. Lazarus. 2005. Optimal system of loops on an orientable surface. *Discr. Comput. Geom.* 33, 3, 507–534.

S. Dong, P.-T. Bremer, M. Garland, V. Pascucci, and J. C. Hart. 2006. Spectral surface quadrangulation. *ACM Trans. Graph.* 25, 3, 1057–1066.

D. Eberly. 1998. Triangulation by ear clipping. http://www.geometrictools.com/Documentation/TriangulationByEarClipping.pdf.

M. Fisher, P. Schroder, M. Desbrun, and H. Hoppe. 2007. Design of tangent vector fields. *ACM Trans. Graph.* 26, 56.

F. Kalberer, M. Nieser, and K. Polthier. 2007. Quadcover – Surface parameterization using branched coverings. http://www.polthier.info/articles/quadCover/KNP07-QuadCover.pdf.

N. Kowalski, F. Ledoux, and P. Frey. 2013. A pde based approach to multidomain partitioning and quadrilateral meshing. In *Proceedings of the 21st International Meshing Roundtable.* Springer, 137–154.

W. C. Li, B. Levy, and J.-C. Paul. 2005. Mesh editing with an embedded network of curves. In *Proceedings of the IEEE International Conference on Shape Modeling and Applications*. 62–71.

D. Martinez, L. Velho, and P. C. Carvalho. 2005. Computing geodesics on triangular meshes. *Comput. Graph.* 29, 5, 667–675.

M. Mrozek. 1995. Conley index theory. In *Handbook of Dynamical Systems II*, Elsevier/North-Holland, 393–460.

A. Myles, N. Pietroni, D. Kovacs, and D. Zorin. 2010. Feature-aligned t-meshes. In *Proceedings of the 37th International Conference and Exhibition on Computer Graphics and Interactive Techniques (SIGGRAPH'10).* 117:1–117:11.

J. Palacios and E. Zhang. 2007. Rotational symmetry field design on surfaces. *ACM Trans. Graph.* 26, 3, 55.

K. Polthier and M. Schmies. 2006. Straightest geodesics on polyhedral surfaces. In *Proceedings of the 33rd International Conference and Exhibition on Computer Graphics and Interactive Techniques (SIGGRAPH-Courses'10).* ACM Press, New York, 30–38.

N. Ray, W. C. Li, B. Levy, A. Sheffer, and P. Alliez. 2006. Periodic global parameterization. *ACM Trans. Graph.* 25, 4, 1460–1485.

N. Ray, B. Vallet, L. Alonso, and B. Levy. 2009. Geometry aware direction field processing. *ACM Trans. Graph.* 29, 1, 1:1–1:11.

N. Ray, B. Vallet, W. C. Li, and B. Levy. 2008. N-symmetry direction field design. *ACM Trans. Graph.* 27, 2, 10:1–10:13.

C. Rossl and H. Theisel. 2012. Streamline embedding for 3d vector field exploration. *IEEE Trans. Vis. Comput. Graph.* 18, 3, 407–420.

B. Spencer, R. S. Laramee, G. Chen, and E. Zhang. 2009. Evenly spaced streamlines for surfaces: An image-based approach. *Comput. Graph. Forum* 28, 6, 1618–1631.

V. Surazhsky, T. Surazhsky, D. Kirsanov, S. J. Gortler, and H. Hoppe. 2005. Fast exact and approximate geodesics on meshes. *ACM Trans. Graph.* 24, 3, 553–560.

A. Szymczak and E. Zhang. 2012. Robust morse decompositions of piecewise constant vector fields. *IEEE Trans. Vis. Comput. Graph.* 18, 938–951.

K. Wang, Weiwei, Y. Tong, M. Desbrun, and P. Schroder. 2006. Edge subdivision schemes and the construction of smooth vector fields. *ACM Trans. Graph.* 25, 3, 1041–1048.

E. Zhang, K. Mischaikow, and G. Turk. 2006. Vector field design on surfaces. *ACM Trans. Graph.* 25, 4, 1294–1326.

Received July 2013; revised December 2013; accepted February 2014