Volume 0 (1981), Number 0 pp. 1–11 COMPUTER GRAPHICS *forum*

Easy Integral Surfaces: A Fast, Quad-based Stream and Path Surface Algorithm

Tony McLoughlin[1], Robert S. Laramee[1], and Eugene Zhang[2]

[1]Visual and Interactive Computing Group
Department of Computer Science, Swansea University, United Kingdom
{*cstony, r.s.laramee*}@swansea.ac.uk

[2]School of Electrical Engineering and Computer Science
Oregon State University, USA
zhange@eecs.oregonstate.edu

Abstract

Despite the clear benefits that stream and path surfaces bring when visualizing 3D vector fields, their use in both industry and for research has not proliferated. This is due, in part, to the complexity of previous construction algorithms. We introduce a novel algorithm for the construction of stream and path surfaces that is fast, simple and does not rely on any complicated data structures or surface parameterization, thus making it suitable for inclusion into any visualization application. We demonstrate the technique on a series of simulation data sets and show that a number of benefits stem naturally from this approach including: easy timelines and timeribbons, easy stream arrows and easy evenly-spaced flow lines. We also introduce a novel interaction tool called a surface painter in order to address the perceptual challenges associated with visualizing 3D flow. The key to our integral surface generation algorithm's simplicity is performing local computations on quad primitives.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.5]: Computational Geometry and Object Modeling—

1. Introduction

Streamlines and pathlines are ubiquitous in flow visualization applications. They are well understood and intuitive tools that produce insightful visualizations. However, these curve-based primitives can suffer greatly from visual complexity, and finding an optimal seeding strategy that produces an uncluttered but detailed visualization is non-trivial. They are also restricted in the characteristics that they represent. For example, a simple tangent line does not exhibit the downstream direction of the underlying vector field.

Stream surfaces are an extension of the streamline. Stream surfaces, surfaces everywhere tangent to the flow, are a viable solution for the visualization of 3D vector fields. Firstly they do not suffer from the visual complexity the same way seeding many streamlines can. Secondly, depth cues can be easily added using shading. The 3D orientation of the surface can clearly be perceived. Path surfaces are the extension of stream surfaces to unsteady flow. They are integral surfaces (computed from integration) everywhere tangent to a time-dependent vector field. Path surfaces provide the same benefits over pathlines as stream surfaces over streamlines.

However, despite the aforementioned benefits integral surfaces provide for the visualization of 3D vector fields, they are seldom used in commercial applications or for research purposes. We hypothesize that this is due to the implementation complexity, and thus inaccessibility, of previous approaches to surface construction.

Simplicity of an algorithm often directly affects its popularity and impact. For example, we believe that much of the success behind the Marching Cubes Algorithm [LC87] stems from the fact that the technique is based on simple, local analysis of cube primitives, thus making the technique highly accessible. This means it can easily be incorporated

into commercial software and it can handle large data sets. It is this kind of accessibility that we strive for here.

In general, previous algorithms are based upon complex data structures and an elaborate mesh triangulation, that rely on several sets of co-ordinates (physical-space, computational-space and parametric-space). This, coupled with the cost of maintaining an optimal triangulation strategy depreciates the attraction of using stream and path surfaces. A re-triangulation may also be necessary, if a global tiling strategy is used, whenever the surface is extended. In contrast, our approach is based on a small set of simple, local operations performed on quad primitives and requires no global re-meshing strategy. Quad-based meshes have become increasingly popular in recent years, and their application to stream and path surfaces seems natural when we consider the anisotropy of flow fields. Quad-based meshes allow for easy extension. When the mesh is extended the previous connectivity remains unchanged and new quads are appended onto the front of the surface. Using a global triangulation would result in the connectivity of the entire mesh having to be recomputed.

The main contribution of this paper is the introduction of a novel algorithm that is significantly easier to model and implement than existing methods with the following benefits:

- An algorithm that is fast because it is based on only a few local operations applied to quad primitives. It requires a simple 2D array and no surface parameterization.
- An algorithm that is suitable for large data sets because of the local nature of the processing.
- An algorithm that supports a number of enhancements that stem naturally from the technique's simplicity including: easy timelines, easy timeribbons, easy stream arrows and easy evenly-spaced streamlines.
- An algorithm that is platform independent and thus widely accessible to researchers and engineers wishing to incorporate it into their vector field visualization application. The algorithm is platform independent because it does not rely on any special graphics hardware. Our implementation utilizes OpenGL 1.2.
- We introduce a novel interaction tool called a stream surface painter designed specifically to address the perceptual problems associated with visualizing 3D flow.

However, in order to achieve these benefits, the same challenges must be addressed as with previous solutions, namely how to construct a smooth and accurate surface in flow characterized by high convergence, divergence and curvature. Although tangent surface construction is fundamentally a vector field sampling problem, this is the first algorithm that formulates the construction of such surfaces explicitly as a sampling problem.

The choice of quad-based meshes is partially inspired by the increase in their popularity within the meshing community [ACSD*03] [TACSD06], and by the following observations:

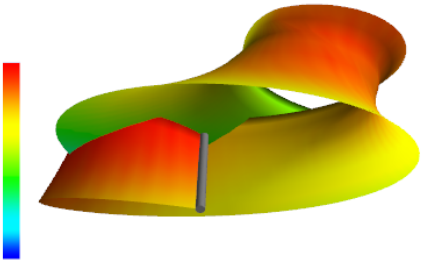

Figure 1: *A quad-based path surface color coded according to velocity magnitude along the core of the tornado from the tornado simulation. This figure shows our algorithm is capable of visualizing a field of high local rotation and twisting behavior. The surface also demonstrates a unique property of path surfaces – the ability to intersect themselves.*

- A quad-based mesh contains edges aligned with the directions of the flow. This results in a more natural placement of the primitives and allows a user to distinguish the flow behavior from the mesh itself, without the need for additional processing. This cannot be said about their triangular counterparts. We draw attention to Figure F in [Hul92] to highlight this.
- Quad meshes make for a high-quality representation of the flow.
- The lines, to some extent, mimic lines that artists themselves might draw when creating depictions of the flow.
- Quads can provide a more compact representation of surface geometry in terms of both space and processing time, e.g., the topology of the mesh does not require explicit storage.

The rest of the paper is organized as follows: Section 2 provides a discussion of the previous work related to stream and path surfaces in the context of our algorithm. Section 3 describes the computational model and the local tests that are performed on the quad primitives. Section 4 shows that several known enhancements to integral surface-based visualizations stem naturally from the quad-based model described in Section 3, namely, easy timelines, easy timeribbons, easy stream arrows, easy evenly-spaced flow lines and the stream surface painter. Section 5 presents some results of the algorithm including its application to a large hurricane simulation. Section 6 concludes the paper and identifies potential directions of future work.

2. Related Work

There have been relatively few proposed solutions to stream and path surface construction, especially when compared to

the volume of effort concentrated on texture-based flow visualization [LHD*04]. A recent survey of integration techniques by McLoughlin et al. provides an overview of most integral surface methods [MLP*09]. Well known work is presented by Hultquist [Hul90,Hul92]. Hultquist's algorithm first involves advancing a front in order to generate streamribbons. During streamribbon front advancement, candidate triangles are generated at each iteration. The goal of this approach is to create a globally-minimal tiling. One disadvantage of creating a globally-minimal tiling is that no triangles can be created until all of the integration of the curves has been completed. Van Wijk [vW93] introduced a method for implicitly generating stream surfaces. This is achieved by defining a continuous function, $f(x,y,z)$, on the boundaries of a flow data set. This function is then extended to the interior of the domain. The iso-contour of the function $f(x,y,z)$ is then used to generate the stream surface. Scheuermann et al. [SBH*01] present a method for the construction of stream surfaces on tetrahedral grids.

Garth et al. [GTS*04] presented an improved method for stream surface construction in areas of intricate flow based on an extension of Hultquist's algorithm. The improved triangulation results from using higher order integration schemes combined with arc length parameterization.

More recently, Schafhitzel et al. introduce an alternative stream surface construction algorithm based on points [STWE07]. However, this approach introduces the new complication of how to generate a smooth and continuous surface from a set of discrete point primitives. Their GPU implementation also places limits on the size of the data sets that can be visualized. With the exception of Schafhitzel et al.'s approach, all previous algorithms are based on complex data structures and surface mesh triangulations. Consequently, they are difficult to implement and use in practice. In contrast, our approach is based on simple, local operations on quad primitives.

Enhancements to stream surfaces are also a fruitful area of research. Laramee et al. [LGSH06] presented an application in which texture advection was applied to stream surfaces to increase the number of characteristics of the flow that could be visualized simultaneously. Stream surfaces are also used to segment a vector field into regions of similar behavior [MBS*04]. This is achieved by growing the surfaces starting near critical points in the vector field. Löffelmann et al. presented stream arrows [LMG97,LMGP97], an enhancement to stream surfaces that removes arrow-shaped holes out of the surface in order to reduce occlusion when the stream surface(s) overlap by allowing the user to see through the displaced sections. This method also serves the purposes of indicating the downstream flow direction by means of the orientation of the arrow. We demonstrate a range of enhancements on top of our core algorithm in Section 4.

The problem of quad-dominant remeshing, i.e., constructing a quad-dominant mesh from an input mesh, has been a well-studied problem in computer graphics. A key observation is that a nice quad-mesh can be generated if the orientations of the mesh elements follow the principle curvature directions [ACSD*03]. This observation has led to a number of efficient remeshing algorithms that are based on streamline tracing [ACSD*03, MK04, DKG05]. Ray et al. [RLL*06] note that better meshes can be generated if the elements are guided by a 4-RoSy field. Dong et al. [DBG*06] perform quad remeshing using spectral analysis, which produces quad meshes that in general do not align with the curvature directions [PZ07]. Felix et al. [KNP07] present a quad-remeshing technique that guarantees no T-junctions. However, this method requires the generation of a surface parameterization on the input mesh, which is not suited for our interactive approach.

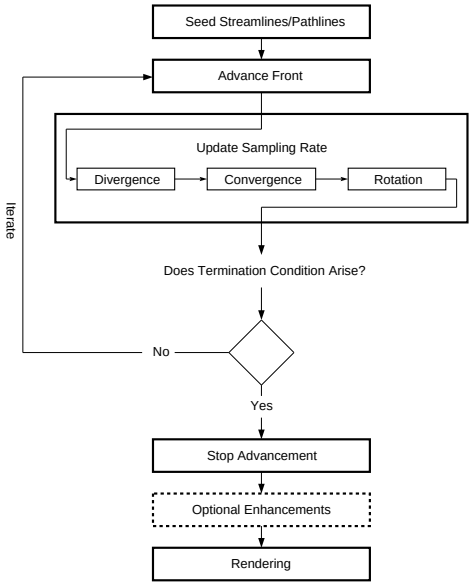

Figure 2: *An overview of the easy integral surfaces algorithm.*

3. Easy Stream and Path Surfaces

Our algorithm consists of a series of operations illustrated in Figure 2:

1. We start out by seeding a curve using an interactive seeding rake (Section 3.1).
2. Then the stream (or path) surface front is advanced according to the vector field (Section 3.1).
3. The next step is to update the sampling rate of the advancing surface front in order to handle divergent, convergent, and rotational flow (Sections 3.2 and 3.3).
4. Terminating conditions such as leaving the domain are tested before returning to step 2 (Section 3.4).
5. User-controlled optional enhancements are enabled and

the scene is rendered according to the current rendering parameters (Section 4).

What follows is a description of each stage of our algorithm along with a description of the technical challenges at each point.

3.1. Seeding and Advancing the Front

We use an interactive seeding curve so that the user can generate a variety of integral surfaces at run time. This also allows the user to quickly investigate any areas of interest within the flow domain. The seeding curve can be placed at any position and the user is able to adjust the length in order to create wider surfaces and analyze more of the vector field in a single visualization. The initial distance between seeds is $\frac{1}{2}d_{sample}$ i.e., $\frac{1}{2}$ the distance between data sample points. However the sampling rate can be adjusted, determining the number of flow lines that are initially seeded along the curve. By flow lines we mean streamlines and pathlines.

We construct an integral surface from quad primitives. There are two important distances to consider when constructing a new quad: (1) d_{sep} - the distance between neighboring flow line points that correspond to the same integration time t and (2) d_a - the advancement distance. Our goal is to generate quads that result in smooth and accurate surfaces. In practice, this is obtained by determining the appropriate lengths of d_{sep} and d_a so that we maintain an appropriate sampling rate of the underlying vector field. The sampling rate we choose in all cases throughout our algorithm is guided by the Nyquist Limit, namely, the sampling frequency must be (at least) twice that of the underlying data frequency for accurate reconstruction. Thus we choose an initial d_{sep}: $d_{sep} < \frac{1}{2}d_{sample}$. Similarly for d_a, we start a new quad when the flow line integration distance exceeds $\frac{1}{2}$ the between data samples. We use a second order Runge-Kutta integrator in our implementation, which provides a good balance between accuracy and computational speed.

Using quads, the advancement of the front is simplified. If triangles are used [Hul90, Hul92, GTS*04] then an optimal tiling strategy should be included in order to prevent long, thin triangles. The approach of Hultquist [Hul90, Hul92] requires three data structures, two tracer structures and a ribbon structure. The two tracer structures are used to generate the sequence of points that define two flow lines S^0 and S^1. Each tracer stores the context required by Hultquist's algorithm to advance a particle through a sampled vector field. The ribbon structure is necessary to connect S^0 and S^1. The position of each point is recorded in physical space, computational space, and the surface parametric space. In contrast our algorithm requires a 2D array of points in physical space only. The two dimensions of the array correlate to the S and t parameters of the surface, thus the mesh topology is stored implicitly within the data structure and does not require explicit computation and storage. To find a quad

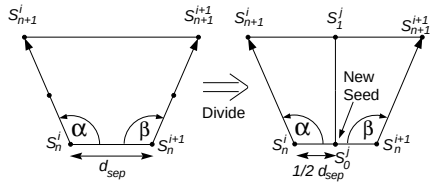

Figure 3: *Divergence is monitored by the testing the value of d_{sep}. When $d_{sep} > \frac{1}{2}d_{sample}$ and $\alpha > 90^\circ$ and $\beta > 90^\circ$ the quad is divided. A new flow line seedpoint S^j_0 is introduced as a result of the subdivision.*

from a given point, we access it's neighboring points in the 2D array. When we encounter four valid points we have a quad. We use the term *valid point* due to the fact that we provide flags indicating the state of a particular vertex for which some cases may be skipped during this search process. These flags and how they are used are discussed later.

3.2. Divergence and Convergence

Divergence is a common phenomenon of flow. When flow diverges the distance between points of neighboring flow lines, d_{sep}, increases. As d_{sep} increases the data sampling rate is reduced. If we were to construct a surface just using these points, the surface could potentially miss critical features of the flow and thus be an inaccurate representation of the underlying field. To overcome this a new flow line is seeded whenever the distance between a pair of flow lines exceeds a threshold. At each stage of the front advancement, the separating edge length between corresponding flow lines is monitored for accurate sampling frequency. As soon as $d_{sep} > \frac{1}{2}d_{sample}$ and $\alpha > 90^\circ$ and $\beta > 90^\circ$ we simply divide the quad as in Figure 3. This results in two quad primitives and a new streamline seeding point. The seed point for the new streamline is calculated by interpolating between the

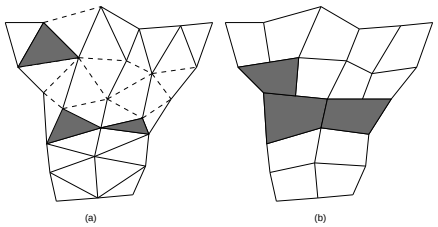

Figure 4: *A comparison between the divergence operations of (a) Hultquist's algorithm and (b) Easy Stream Surfaces divergence operation. The grey colored sections indicate the mesh transition.*

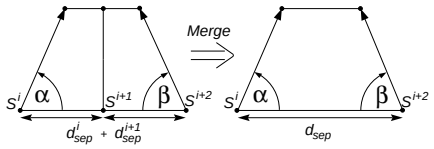

Figure 5: *Convergence is detected when $d^i_{sep} + d^{i+1}_{sep} < \frac{1}{2} d_{sample}$ and $\alpha < 90^\circ$ and $\beta < 90^\circ$. It is handled by terminating the middle flow line. Two quad primitives are merged into a single quad.*

distance halfway between S^i_n and S^{i+1}_n. Using quads results in a much simpler mesh, see Figure 4.

In contrast, the approach of Hultquist [Hul90, Hul92] requires new tracer and ribbon structures to be stored in the linked list which represents the front. Also previous methods describe *how* new particles are inserted in order to handle convergence (and divergence) but not *when*. By formulating the algorithm as a sampling problem explicitly, the answer as to when new particles should be inserted (or removed) from the flow becomes clear.

In order to add new streamlines into our 2D array and maintain the topology information, we have to move columns over in order to accommodate the new streamline. However, this results in elements within these newly inserted columns that don't contain a mesh vertex. We handle this by having a flag that indicates whether a point exists at a particular array element or whether it contains no vertex information. Elements flagged as empty are simply skipped over during the rendering process and by the sampling rate operations. Instead, their next valid neighbor is used (as these are the correct neighbors in physical-space). Note that there

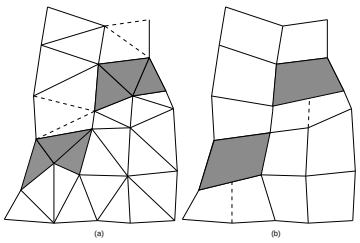

Figure 6: *A comparison between the merging operations of (a) Hultquist's algorithm and (b) Easy Stream Surfaces merging operation. The grey colored sections indicate the transition using the convergence operation. In (a) three new triangles must be created, while in (b) no extra primitives have to be inserted in order to create a smooth transition between the strip widths.*

are more flags defined for the elements, these are discussed where they are appropriate in future sections.

A vector field may also locally converge. Adjacent flow lines begin to approach each other causing the distance between the corresponding points of S^0 and S^1 to decrease. This can lead to many points being generated in a small vicinity on the stream surface. This results in an area being oversampled and, very small quads being produced for the surface mesh. We monitor for cases of flow convergence at each stage of the surface front advancement. As soon as $d^i_{sep} + d^{i+1}_{sep} < \frac{1}{2} d_{sample}$ and $\alpha < 90^\circ$ and $\beta < 90^\circ$ we simply terminate streamline S^{i+1}, the flow line between S^i and S^{i+2}. The result is a quad merging (Figure 5). Hultquist's algorithm [Hul90, Hul92] requires the insertion of three new triangles which make the transition between the leading edges of two ribbons and the single leading edge of a new replacement. We point out that, once again, quad meshes result in a simpler mesh, see Figure 6. Note, no description of when adjoining ribbons are merged into one is given (only how).

Quad meshes can introduce T-junctions which may cause gaps. For divergence, no gaps are produced as S^j_0 is chosen to lie on the edge from S^j_n to S^{j+1}_n. However, during convergence, gaps may occur. This can be handled in two different ways. One is by dragging S^{i+1} onto edge $S^i - S^{i+2}$ for the adjacent quads. Another solution is to simply patch the gap with a triangle. Our framework supports both solutions. We also point out that the vast majority of cases are divergent.

3.3. Curvature

After advancing the front by one quad, we monitor the flow for high curvature. Under curving flow we want the edges of our quad primitive to maintain their basic shape. One of the elements of our algorithm is ensuring that after each integration step the local advancing front remains generally orthogonal to the downstream direction. This maintains a quad mesh with desirable properties. Curvature within the flow field challenges the goal of maintaining edge shape. When flow lines are integrated through curving flow, the flow lines

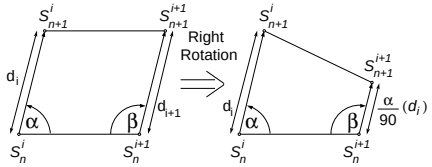

Figure 7: *Given a vector field under local right rotation, where $\alpha < 90^\circ$ and $\beta > 90^\circ$ the integration step-size between S^{i+1}_n and S^{i+1}_{n+1} is decreased by a factor proportional to the amount of rotation. Likewise for the case of left rotational flow.*

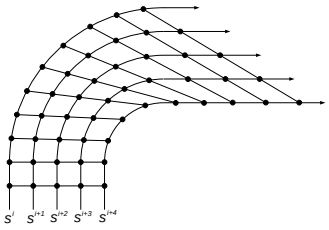

Figure 8: *Curvature in the flow field deforms the orthogonal tangent surface front and results in sheared quad primitives for the underlying surface mesh.*

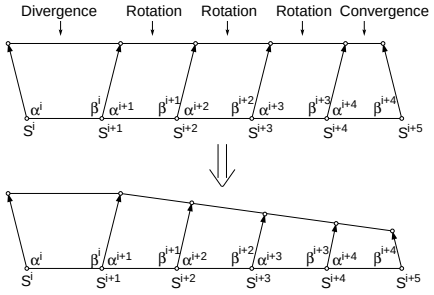

Figure 9: *In order to generate a smooth front of quads under rotation we group them and process them from left-to-right for the case of right rotation. Similarly for the case of left rotation.*

on the inside of the curvature follow a shorter path and the flow line front does not remain orthogonal to the flow direction, see Figure 8. This results in the underlying quads becoming sheared and warped. This gets progressively worse. Also the more rapid the curvature the more severe the shearing of the quad primitives.

To overcome this we apply a method suggested by Darmofal and Haimes [DH92] and presented in detail by Kenwright and Lane [KL95]. We use an adaptive step size integrator that measures the angle between velocity vectors and the quad base at each stage of the front advancement in areas of flow curvature. To produce an optimal result, the integration step size must be a function of the length on the adjacent streamline. We look at the interior angles of the quads in order to adjust the quads edge lengths as in Figure 7, which shows the case of a quad under right-rotational flow ($\alpha < 90^\circ$ and $\beta > 90^\circ$). We adjust the step-size between S_n^{i+1} and S_{n+1}^{i+1} such that the quad can maintain its shape (and accuracy) under rotational flow. In this case d_{i+1} is shortened proportional to the angle α using the relation $d_{i+1} = \frac{\alpha}{90} d_i$, where d_i is the step-size between S_n^{i+1} and S_{n+1}^{i+1}, d_{i+1} is the step-size between S_n^{i+1} and S_{n+1}^{i+1}. In order to avoid a jagged surface front under flow rotation we process rows of quads successively as a unit with common flow behavior. We need to ensure that the flow lines are adjusted in the correct order. Our stream and path surface front advancement uses two passes as shown in Figure 9. In the first pass we process the divergent and convergent quads and simply identify the quads under rotation. This results in a grouping of quads according to common flow behavior, e.g, a region of right-rotating flow.

For a strip of right-rotating quads we process them by marching from left-to-right order adjusting the edge lengths as shown in Figure 9. For left-rotating flow we perform the mirror opposite. We then iterate along each group adjusting the step-size of the latest segment. If flow starts to diverge we get two major groupings of flow lines: one group curving in each direction. The grouping method produces reasonable results with two fronts that expand out, one for each direction of rotation. In the extreme case of coinciding convergent and rotating quads, e.g. $\alpha << 90^\circ$, $\beta > 90^\circ$ and $d_{sep}^i + d_{sep}^{i+1} < \frac{1}{2} d_{sample}$ under right rotational flow, we first perform a merge followed by a rotation operation. Similarly for extreme cases of coinciding divergent and rotating quads. We note that no description of how to handle rotating flow is provided in previous literature [Hul90, Hul92, GTS*04].

3.4. Splitting and Termination

If an object is encountered in the flow field, the surface can split into two sections that advance in separate directions.

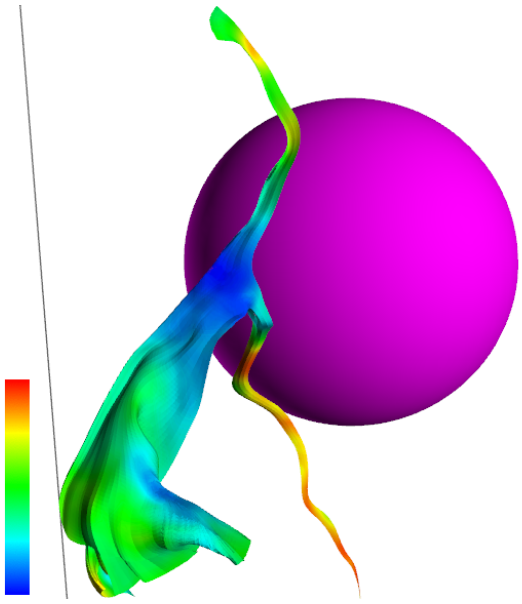

Figure 10: *When an object boundary is encountered the surface splits. The surface is torn and the separate portions are advanced independently of each other. Color is mapped to velocity magnitude.*

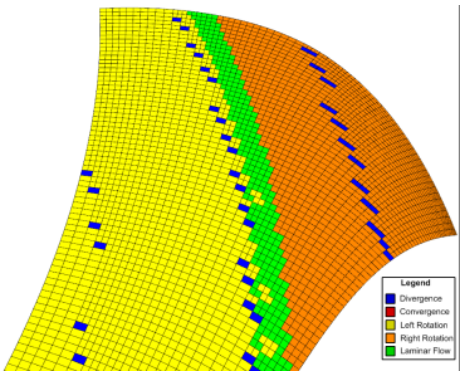

Figure 11: *Our algorithm handles widely diverging flow while maintaining the desired organized advancing front. This surface is colored according to the underlying flow characteristics: left rotation is mapped to yellow, right rotation is mapped to orange, laminar flow is mapped to green, divergence is mapped to blue, convergence is mapped to red.*

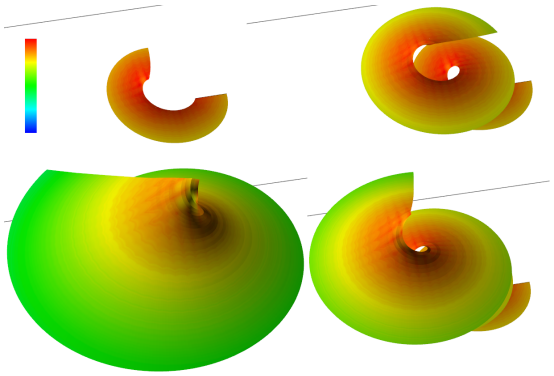

Figure 12: *The stream surface painter shows the evolution of the construction of the stream surface. Starting from the top-left image and commencing clockwise the stream surface painter has been set to render 25%, 50%, 75% and 100% of the stream surface respectively. The stream surface is colored according to the vector field magnitude.*

This is handled by terminating the flow lines contained in the surface that encounters a boundary (or other terminating condition). The sections on either side of terminated flow lines are then calculated independently. This prevents the whole surface advancement being halted unnecessarily and ensures that the separating portions maintain their accuracy and a well structured mesh. Figure 10 shows a stream surface splitting and the two sections advancing independently.

To handle splitting we introduce a new flag for the vertices. This indicates that a split has occurred. If we encounter a split flag while searching for the neighboring vertices, we simply halt the search for the vertices as we are on the edge of the surface. We then move onto the next valid point and search for it's neighbors that comprise a quad. This allows multiple sections of the surface to be computed independently of each other.

Surface advancement is terminated when one of a series of termination conditions is met. The termination conditions are: the entire surface front leaves the vector field domain, the distance that the surface has traveled exceeds a preset geodesic length, or an area of zero velocity is encountered, i.e a solid object within the flow field or a critical point.

4. Enhancements

A simpler surface generation scheme is not the only benefit of our algorithm. In this section we demonstrate that several enhancements stem naturally from its quad-centric point of view.

4.1. Surface Painter

The first enhancement we discuss is called the surface painter. The surface painter is used to adjust the geodesic length of the surface (from the seeding rake to the surface front). Using a slider the user is able to interactively "paint" the surface in the downstream (or upstream) direction of the flow. This allows the user to easily observe how the surface evolves. It also addresses the challenge of occlusion resulting from overlapping portions of the surface, see Figure 12. With the interactive surface painter control the viewer can "rewind" the surface thereby revealing what would otherwise be occluded portions of the surface. The user can simply paint the surface automatically and view an animation of the surface front evolving step-by-step. A video demonstration of the surface painter is provided [MLZ].

Although such an interaction would be possible with previous methods [GTS*04, Hul90, Hul92] it would require a mesh parameterization. Previous methods require a parameterization of both the displacement of points along the streamlines ($s \in [0,1]$) and the advected images of the seed rake along the downstream displacements ($t \in [0,N]$). A parametric description of the surface is unnecessary with our method.

4.2. Easy Timelines and Easy Timeribbons

A timeline is the line formed by connecting a series of massless particles in the flow seeded at the same instant of time (but at different locations). Although they are often mentioned as a very useful vector field visualization technique (because they show the convergent and divergent behavior of the flow), they are not commonly featured in a visualization software application. This may be due to complex implementation challenges. Timelines can easily be implemented from our algorithm by using the stream surface front at each integration stage and simply connecting these points.

Two minor adjustments have to be made to the surface

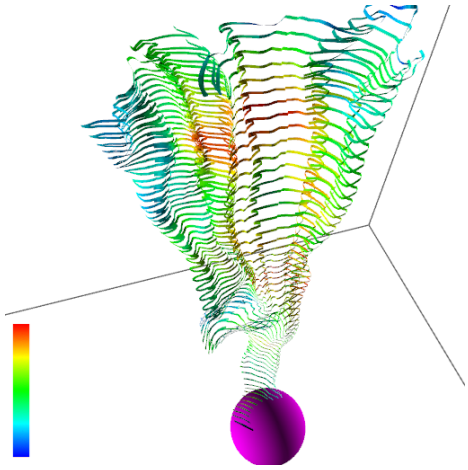

Figure 13: *This image shows timeribbons rendered as shaded strips of quads to visualize a smoke plume simulation. Color is mapped to local velocity magnitude.*

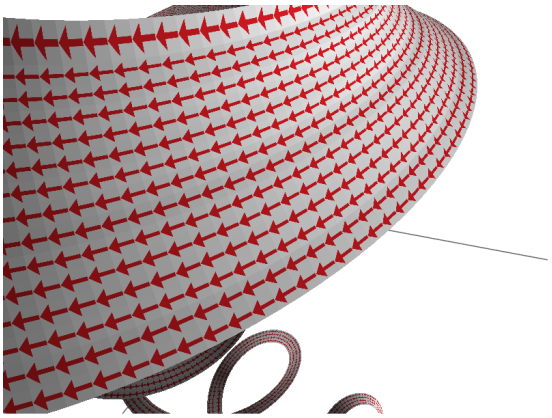

Figure 14: *An arrow texture can simply be mapped to the quad primitives to show the downstream direction of flow. The arrows can then be animated by scrolling the texture [MLZ].*

construction algorithm in order to accommodate timelines. We use a flow line integrator whose step-size is proportional to velocity magnitude (as opposed to using an integrator with a uniform step-size). We also need to disable the rotation operations, described in Section 3.3.

The result are timelines that show the range of speed at which different regions of the flow travel. The advancing timelines benefit from the changes in sampling frequency in accordance with the divergence and convergence operations. Animating the timelines is a simple process that may produce insightful visualizations of the evolution of the flow [MLZ]. We extend timelines using quads as opposed to using line primitives, the result are a novel geometric visualization called *timeribbons*. Just as streamribbons extend streamlines to show curling flow behavior, timeribbons extend timelines to show oscillating (or wave-like) flow behavior, see Figure 13. We simply use a row of quads and render every n^{th} row. This forms ribbons that are easier to observe in a 3D scene by providing the perceptual benefits of using polygonal primitives over lines, e.g. shading. A video demonstration of animated timelines and timeribbons created by our application is provided [MLZ]. Timelines and timeribbons would be possible to incorporate into previous algorithms [GTS*04, Hul90, Hul92]. A surface parameterization, $S \in [0,1]$, $t \in [0, T_{max}]$, is required whereas timelines and timeribbons are a natural byproduct of our approach.

4.3. Easy Stream and Path Arrows

Stream arrows are arrow-separated portions of a stream surface. They enhance the basic stream surface visualization in (a minimum of) three ways: (1) The direction of the arrows shows the downstream direction of the flow. (2) The arrows (or their complement with respect to the stream surface) can be made transparent thus allowing the viewer to see through portions of the surface. This alleviates the occlusion problems caused by overlapping portions of the geometry. (3) They enrich wide stream surfaces with internal visual structure. Löffellmann et al. [LMG97,LMGP97] present two algorithms for the placement of stream arrows on stream surfaces. They involve the use of surface data structures that enable fast searching of the geometry in order to find a suitable place to map texture-based arrows.

Our mesh construction algorithm allows us to readily map an arrow texture onto the surface. The regular pattern generated by the quad primitives is ideal for tiling a texture: it requires no additional data structures and no search process as in previous approaches. It is also straight forward to add a user option that allows for the textures to cover several quads, thus increasing the size of the textured arrows in line with user preference and suitability of the current rendering. See Figure 14 for example results.

Animation of the arrows, can be added. This provides another insightful enhancement in which to visualize the flow characteristics of the surface. A video demonstration of animated stream arrows is provided [MLZ]. Stream and path arrows are more difficult to place if a triangular mesh is used.

4.4. Easy Evenly-Spaced Flow Lines

A range of algorithms have been proposed for the creation of evenly-spaced streamlines [JL97, LS07, LM06, TB96]. Evenly-spaced streamlines or pathlines clearly capture the features of the flow field by distributing the streamlines or pathlines evenly on the surface. They also produce illustrative visualizations similar to those found in textbook depictions of hand-drawn flow. Note that evenly-spaced seeds do

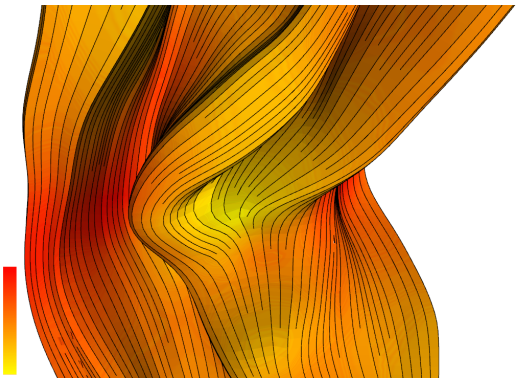

Figure 15: *We can create a hybrid visualization of surfaces and evenly-spaced flow lines. Color is mapped to velocity magnitude.*

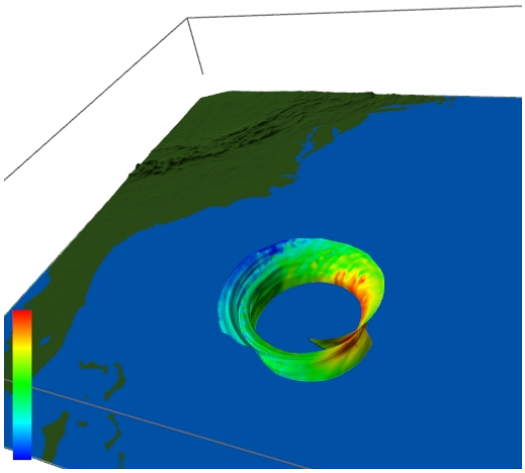

Figure 16: *A stream surface depicting the eye of Hurricane Isabel.*

not necessarily result in evenly-spaced streamlines. Evenly-spaced flow lines are a simple by-product of our surface construction. The merge and divide operations ensure that $d_{sep} \approx \frac{1}{2} d_{sample}$. In fact, the d_{sep} parameter used throughout this paper is very similar to the d_{sep} parameter used by Jobard and Lefer [JL97]. Figure 15 shows how our easy integral surface algorithm can be used for the depiction of evenly-spaced streamlines. Evenly-spaced streamlines are not such an obvious by-product with triangular meshes.

5. Results

In practice our criterion for right rotation, $\alpha > 90^\circ$ and $\beta > 90^\circ$, proves too rigid. In the implementation we use a more relaxed constraint: $(\alpha > 90^\circ + \varepsilon_{rotation})$ and $(\beta < 90^\circ - \varepsilon_{rotation})$ where $\varepsilon_{rotation}$ is a user-defined threshold. We found a value of $\varepsilon_{rotation} = 3^\circ$ yields good results. In extreme cases of divergence, if $d_{sep} >> \frac{1}{2} d_{sample}$, i.e $d_{sep} = n \cdot d_{sample}$ after advancing the front we can divide the quad and insert multiple new seeding points at a rate of $2n$, where $n = \frac{d_{sep}}{d_{sample}}$.

We also note the potential wasted memory in the 2D array. Elements marked as empty or split contain no explicit information with respect to the surface geometry. However, a mesh containing 1,000,000 vertices uses less then 16MB of memory. Compared to the large amounts of RAM in a typical PC nowadays (usually in the order of GB) that this is a minor issue in order to pursue an much simpler implementation.

Figure 16 shows stream surfaces used to visualize the Hurricane Isabel data set. One seeding curve has been placed in order to depict the eye of the storm. The algorithm was run on the original $500^2 \times 100$ resolution simulation. The local processing nature of our algorithm enables fast processing with large data sets. Figure 1 shows a path surface generated on the tornado data set of size 128^3. This stream surface was seeded close to the core of the tornado. The easy integral surface algorithm was implemented in C++ on a PC with a 2.66Ghz Intel Core 2 Duo processor with 4GB RAM and a 256MB nVidia GeForce 8600GT graphics card. We tested 3 different sizes of stream surface, consisting of 10,000, 50,000 and 100,000 quads. Two different integrators were compared for each size of stream surface, namely an Euler integrator and a second-order Runge Kutta integrator. The results of these tests are presented in Table 5, three different size surfaces were created for each integrator, consisting of 10,000, 50,000 and 100,000 quads. The algorithm is fast even though it does not rely on any special-purpose GPU programming. In our review of the previous literature, the last report of performance times for polygonal stream surface construction was Hultquist [Hul90] in 1992, thus making performance time comparisons with previous work non-trivial. We note that the quad mesh lends itself to easy mesh simplification as a post-processing step for large meshes with smooth regions. See Daniels et al [DSSC08] for examples.

	Stream surface size		
	10k quads	50k quads	100k quads
Euler	0.01s	0.04s	0.08s
RK2	0.02s	0.06s	0.12s

Table 1: *This table shows a range of stream surface construction times measured in seconds using our algorithm. Both an Euler and a second order Runge Kutta integrator are compared.*

6. Conclusion

We present a novel integral surface construction algorithm whose simplicity readily allows for implementation and incorporation into visualization applications. The techniques benefits from the advances that quad-based approaches. The algorithm is fast and handles large high-resolution datasets due to the local nature of its processing. The algorithm comprises of operations that monitor local flow for characteristics such as rotation, convergence and divergence and does not require a parameterization of the surface. Our formulation is given explicitly in terms of data sampling thus making decisions regarding when and where to insert or delete new flow lines clear. The enhancements presented are easy stream and path arrows, easy timelines, easy timeribbons and easy evenly-spaced flow lines. We also presented the surface painter. All enhancements stem more naturally from quad meshes as opposed to triangular meshes. The implementation is CPU-based and uses OpenGL 1.2, thus making it widely portable. The availability of an efficient, simple-to-implement integral surface construction algorithm, which is immediately open to useful enhancements could, we believe, have a significant impact on the adoption of tangent surfaces as a visualization tool.

Future work includes a GPU-based version of the algorithm.

Acknowledgments

This work was supported in part by the EPSRC Research Grant EP/F002335/1. Eugene Zhang was partially supported by NSF CCF-0546881. The authors would like to thank Roger Crawfis for providing the procedural function used to generate the tornado data set [CM93] and Ben Spencer for providing the fluid solver used to generate the smoke plume data set. The authors would like to thank Guoning Chen and George Buchanon for valuable feedback and proofreading. Hurricane Isabel data produced by the Weather Research and Forecast (WRF) model, courtesy of NCAR and the U.S. National Science Foundation (NSF).

References

[ACSD*03] ALLIEZ P., COHEN-STEINER D., DEVILLERS O., LÉVY B., DESBRUN M.: Anisotropic polygonal remeshing. *SIGGRAPH 03 - ACM Transactions on Graphics 22*, 3 (2003), 485–493.

[CM93] CRAWFIS R. A., MAX N.: Texture Splats for 3D Scalar and Vector Field Visualization. In *Proceedings IEEE Visualization '93* (Oct. 1993), IEEE Computer Society, pp. 261–267.

[DBG*06] DONG S., BREMER P.-T., GARLAND M., PASCUCCI V., HART J. C.: Spectral surface quadrangulation. *ACM Trans. Graph. 25*, 3 (2006), 1057–1066.

[DH92] DARMOFAL D., HAIMES R.: *Visualization of 3-D Vector Fields: Variations on a Stream*. Paper 92-0074, AIAA, 1992.

[DKG05] DONG S., KIRCHER S., GARLAND M.: Harmonic functions for quadrilateral remeshing of arbitrary manifolds. *Comput. Aided Geom. Des. 22*, 5 (2005), 392–423.

[DSSC08] DANIELS J., SILVA C. T., SHEPHERD J., COHEN E.: Quadrilateral mesh simplification. In *Proc of Siggraph Asia* (2008), p. forthcoming.

[GTS*04] GARTH C., TRICOCHE X., SALZBRUNN T., BOBACH T., SCHEUERMANN G.: Surface Techniques for Vortex Visualization. In *Data Visualization, Proceedings of the 6th Joint IEEE TCVG–EUROGRAPHICS Symposium on Visualization (VisSym 2004)* (May 2004), pp. 155–164.

[Hul90] HULTQUIST J. P. M.: Interactive Numerical Flow Visualization Using Stream Surfaces. *Computing Systems in Engineering 1*, 2-4 (1990), 349–353.

[Hul92] HULTQUIST J. P. M.: Constructing Stream Surfaces in Steady 3D Vector Fields. In *Proceedings IEEE Visualization '92* (1992), pp. 171–178.

[JL97] JOBARD B., LEFER W.: Creating Evenly–Spaced Streamlines of Arbitrary Density. In *Proceedings of the Eurographics Workshop on Visualization in Scientific Computing '97* (1997), vol. 7, pp. 45–55.

[KL95] KENWRIGHT D. N., LANE D. A.: Optimization of Time-Dependent Particle Tracing Using Tetrahedral Decomposition. In *Proceedings of IEEE Visualization 1995* (1995), pp. 321–328.

[KNP07] KÄLBERER F., NIESER M., POLTHIER K.: Quadcover - surface parameterization using branched coverings. *Computer Graphics Forum 26*, 3 (2007), 375–384.

[LC87] LORENSEN W. E., CLINE H. E.: Marching Cubes: a High Resolution 3D Surface Construction Algorithm. In *Computer Graphics (Proceedings of ACM SIGGRAPH 87, Anaheim, CA)* (July 27–31 1987), ACM, pp. 163–170.

[LGSH06] LARAMEE R. S., GARTH C., SCHNEIDER J., HAUSER H.: Texture-Advection on Stream Surfaces: A Novel Hybrid Visualization Applied to CFD Results. In *Data Visualization, The Joint Eurographics-IEEE VGTC Symposium on Visualization (EuroVis 2006)* (2006), Eurographics Association, pp. 155–162,368.

[LHD*04] LARAMEE R. S., HAUSER H., DOLEISCH H., POST F. H., VROLIJK B., WEISKOPF D.: The State of the Art in Flow Visualization: Dense and Texture-Based Techniques. *Computer Graphics Forum 23*, 2 (June 2004), 203–221.

[LM06] LIU Z. P., MOORHEAD, II R. J.: An Advanced Evenly-Spaced Streamline Placement Algorithm. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (sep/oct 2006), 965–972.

[LMG97] LÖFFELMANN H., MROZ L., GRÖLLER E.: Hierarchical Streamarrows for the Visualization of Dynamical Systems. In *Visualization in Scientific Computing '97* (1997), Eurographics, Springer-Verlag, pp. 155–164.

[LMGP97] LÖFFELMANN H., MROZ L., GRÖLLER E., PURGATHOFER W.: Stream Arrows: Enhancing the Use of Stream-surfaces for the Visualization of Dynamical Systems. *The Visual Computer 13* (1997), 359–369.

[LS07] LI L., SHEN H.-W.: Image-Based Streamline Generation and Rendering. *IEEE Transactions on Visualization and Computer Graphics 13*, 3 (2007), 630–640.

[MBS*04] MAHROUS K., BENNETT J. C., SCHEUERMANN G., HAMANN B., JOY K. I.: Topological segmentation in three-dimensional vector fields. *IEEE Transactions on Visualization and Computer Graphics 10(2)* (2004), 198–205.

[MK04] MARINOV M., KOBBELT L.: Direct anisotropic quad-dominant remeshing. In *PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 207–216.

[MLP*09] MCLOUGHLIN T., LARAMEE R. S., PEIKERT R., POST F. H., CHEN M.: Over Two Decades of Integration-Based, Geometric Flow Visualization. In *Eurographics 2009: State-of-the-Art Reports* (30 March – 3 April 2009), p. To Appear.

[MLZ] MCLOUGHLIN T., LARAMEE R. S., ZHANG E.: Easy stream surfaces: Supplementary video accompanied with paper submission. http://cs.swan.ac.uk/∼cstony/Video/EIS.mpg.

[PZ07] PALACIOS J., ZHANG E.: Rotational symmetry field design on surfaces. *ACM Transactions on Graphics 26*, 3 (2007), 55.

[RLL*06] RAY N., LI W. C., LÉVY B., SHEFFER A., ALLIEZ P.: Periodic global parameterization. *ACM Trans. Graph. 25*, 4 (2006), 1460–1485.

[SBH*01] SCHEUERMANN G., BOBACH T., HAGEN H., MAHROUS K., HAMANN B., JOY K. I., KOLLMANN W.: A Tetrahedral-based Stream Surface Algorithm. In *Proceedings IEEE Visualization 2001* (Oct. 2001), pp. 151–157.

[STWE07] SCHAFHITZEL T., TEJADA E., WEISKOPF D., ERTL T.: Point-Based Stream Surfaces and Path Surfaces. In *GI '07: Proceedings of Graphics Interface 2007* (New York, NY, USA, 2007), ACM, pp. 289–296.

[TACSD06] TONG Y., ALLIEZ P., COHEN-STEINER D., DESBRUN M.: Designing quadrangulations with discrete harmonic forms. In *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing* (Aire-la-Ville, Switzerland, Switzerland, 2006), Eurographics Association, pp. 201–210.

[TB96] TURK G., BANKS D.: Image-Guided Streamline Placement. In *ACM SIGGRAPH 96 Conference Proceedings* (Aug. 1996), pp. 453–460.

[vW93] VAN WIJK J. J.: Implicit Stream Surfaces. In *Proceedings of Visualization '93* (1993), pp. 245–252.