Constructing Streak Surfaces for 3D Unsteady Vector Fields

Tony McLoughlin* Swansea University
Robert S. Laramee† Swansea University
Eugene Zhang‡ Oregon State University

Abstract

Visualization of 3D, unsteady flow (4D) is very difficult due to both perceptual challenges and the large size of 4D vector field data. One approach to this challenge is to use integral surfaces to visualize the 4D properties of the field. However the construction of streak surfaces has remained elusive due to problems stemming from expensive computation and complex meshing schemes. We present a novel streak surface construction algorithm that generates the surface using a quadrangular mesh. In contrast to previous approaches the algorithm offers a combination of speed for exploration of 3D unsteady flow, high precision, and places less restriction on data or mesh size due to its CPU-based implementation compared to a GPU-based method. The algorithm can be applied to large data sets because it is based on local operations performed on the quad primitives. We demonstrate the technique on a variety of 3D, unsteady simulation data sets to show its speed and robustness. We also present both a detailed implementation and a performance evaluation. We show that a technique based on quad meshes handles large data sets and can achieve interactive frame rates.

CR Categories: Computer Graphics [I.3.3]: Picture/Image Generation—Line and curve generation

Keywords: Flow Visualization, Unsteady Vector Fields

1 Introduction

Streaklines are a visualization technique commonly used to depict complex, time-varying phenomena. A streakline is the line joining a series of massless particles passing through a common point at distinct successive times. A streakline corresponds to the use of dye injection from a fixed point source in laboratory flow visualization.

Streak surfaces are the culmination of continuously seeding a curve from a fixed spatial location over time. However, despite the advantages and insight that streak surfaces enable when investigating flow fields, they are not included into visualization applications. This is due to the computational complexity involved in generating their dynamic meshes. The computational cost of constructing streak surfaces stems from factors including the large number of integrations required. In contrast to stream and path surfaces, *every* vertex in the mesh must be integrated at *each* time step. For large meshes this results in a large number of computations per time step. Another major source of computational expense arises from computing the connectivity of the mesh vertices. Divergence, convergence and shear can occur anywhere in the surface at any time, not just the surface front. Due to the time-dependent nature of a streak surface a re-triangulation of the surface may be necessary after every integration step in order to avoid long, thin triangles resulting from divergence, convergence, and shear. These factors generally make the inclusion of streak surfaces into visualization packages prohibitively expensive. Previous streak surfaces algorithms are either (1) GPU-based: very fast but place more restrictions on mesh size and precision than a CPU-based version or (2) CPU-based but not fast enough to support interactive exploration of the flow.

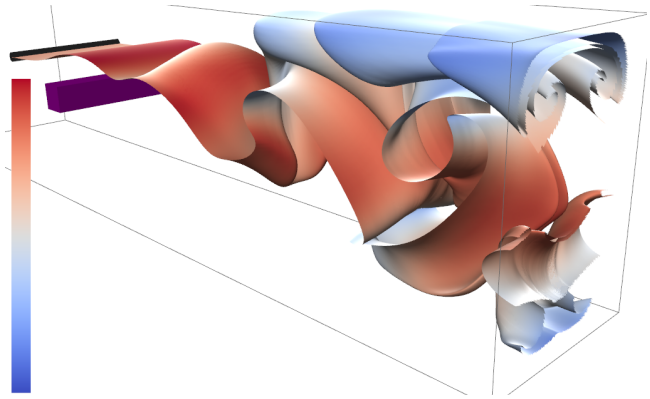

Figure 1: A complete streak surface depicting flow past a cuboid computed and rendered at approximately 3 frames-per-second. This image shows complex regions illustrating interesting flow structures. Color is mapped to velocity magnitude. The full animation is given in the supplementary video.

We present a novel CPU-based streak surface algorithm (Figure **??**) that can both run at interactive frame rates and places kess restriction on mesh size and precision than a gpu-based implementation. The main contributions of this paper are:

- The introduction of a novel streak surface algorithm that can offer both interactive frame rates and high precision.
- An algorithm that is suitable for use on large out-of-core data sets and relatively simple to implement as a result of the local operations performed on quads. Thus, it is suitable for inclusion in any visualization system.
- A streak surface model and implementation for the explicit treatment of shear flow.

We provide platform-independent, CPU-based pseudo-code in order to facilitate implementation into any visualization application. We provide user-controlled parameters that allow the user to trade off between performance and accuracy. This enables the user to switch between modes for quick investigation of the flow domain and high-accuracy representation for presentation and analysis.

The use of quad meshes has increased in recent years, with many algorithms aimed at quad-based re-meshing or simplification of quad-meshes. Some benefits of quad-based meshes are demonstrated by Alliez et al. [Alliez et al. 2003] and Tong et al. [Tong et al. 2006].

However in order to generate such an algorithm several challenges must be overcome such as maintaining a continuous, accurate, quad mesh under flow convergence, divergence and shear. The rest of the paper is organised as follows: Section 2 provides a discussion

*e-mail: cstony@swan.ac.uk
†e-mail:R.S.Laramee@swan.ac.uk
‡e-mail:zhange@eecs.oregonstate.edu

of previous work related to flow surface construction algorithms. Section 3 describes the computational model of our algorithm. A detailed discussion of the implementation is provided in Section 4. Section 5 presents an evaluation of the algorithm showing it applied to various simulation data sets. Finally, Section 6 concludes the paper and identifies areas of future research.

2 Related Work

See McLoughlin et al [McLoughlin et al. 2009a] [McLoughlin et al. 2010] for a complete overview of geometric visualization techniques. Much effort has been focused on construction of *stream surfaces* – surfaces everywhere tangent to a steady-state vector field. Hultquist introduced a method based on an advancing front [Hultquist 1992]. The sampling rate is adjusted by the insertion or removal of streamlines. In contrast to the local method of stream surface presented by Hultquist [Hultquist 1992], Van Wijk presents a global approach for stream surface generation [van Wijk 1993]. A continuous function $f(x,y,z)$ is placed on the boundaries of the data set. An iso-surface extraction technique can then be used to construct the stream surface. Scheuermann et al. devised a method where the underlying tetrahedral grid is used in the construction of the stream surface [Scheuermann et al. 2001]. Garth et al. [Garth et al. 2004] present a method for the construction of stream surfaces in areas of complex flow. This is based upon the advancing front method introduced by Hultquist [Hultquist 1992]. Laramee et al. [Laramee et al. 2006] combined texture advection with stream surfaces to provide a more detailed visualization by showing the inner flow structure of the surface. All of these previous methods are restricted to steady-state flow.

A point-based method for *stream* surface and *path* surface construction was introduced by Schafhitzel et el. [Schafhitzel et al. 2007]. Insertion and removal of points are handled similar to Hultquist's method [Hultquist 1992] to maintain sufficient density of points in order to create an enclosed surface when they are rendered using point-sprites.

Garth et al. present an improved *stream* and *path* surface construction algorithm focusing on high accuracy [Garth et al. 2008]. This method decouples the surface integration and the surface rendering process. The surface construction comprises of advecting a set of timelines through the flow field. Connecting curves representing timelines are then computed. These curves are subject to given predicates in order to refine them where necessary.

McLoughlin et al. [McLoughlin et al. 2009b] demonstrate a simplified stream surface construction method using quad primitives. The refinement of the surface front is performed on a quad-by-quad basis. Quads may be split or merged to maintain sufficient sampling of the vector field. Shearing is handled by analyzing the surface front and processing groups of quads to make them more regular. This method produces an implicit parameterization of the mesh, allowing for a series of enhancements (such as stream arrows) to be included.

Schneider et al. [Schneider et al. 2009] present a method of stream surface construction using a higher-order interpolation scheme. This method provides very smooth surfaces of fourth-order accuracy.

It is important to note that all of the above approaches focus on stream surfaces and/or path surfaces, whereas our work focuses on streak surfaces.

Von Funck et al. [von Funck et al. 2008] describes the construction of *smoke* surfaces. Smoke surface generation involves coupling the opacity of the triangles that comprise the mesh to their size and shape. The more the interior angles of the triangle deviate from 60° the more transparent the surface becomes. Mesh re-triangulation is avoided and this produces a surface approximating the smoke optical model. However, areas of complex flow become transparent by definition, thus interesting features may not be visible.

Krishnan et al. [Krishnan et al. 2009] present a novel streak and time surface algorithm. This technique guarantees a C^1 continuous curve for the integral curves. Three basic operations are defined for the surface adaptation process, these are *edge split*, *edge flip* and *edge collapse*. An edge split ensures that no edge on a triangle is longer than a prescribed threshold. Edge flipping locally refines an area to maximize the minimum angles within the triangles such that triangles are more regular. An edge collapse removes edges from the mesh in regions where the density of triangles is too high. They present a CPU-based implementation for unstructured meshes that runs on the order of hours, thus it does not support interactive visualization of the flow.

Bürger et al. [Bürger et al. 2009] present two streak surface techniques implemented on the GPU. The first technique is based on quads. Each quadrilateral patch contains four vertices. The same vertex is stored (and propagated) multiple times. Refinement of patches is achieved by splitting the longest edge of the quadrilateral and the edge opposite it. This may result in discontinuities within the mesh. This is resolved during the rendering phase where the patch vertices are extended along the line that passes through the centroid and the vertex. A two-pass rendering operation ensures that the quads form a smooth surface during the rendering phase. Both the quad and triangle-based versions of the algorithm are very fast, e.g., serveral frames per second. However their GPU-implementation places unnecessary limits on both the size of the streak surface and the data set: All of which need to fit into GPU memory. Also, shear flow is not handled.

We present the first streak surface algorithm that combines the properties of speed, quality and size. It runs at fast frame rates supporting exploration of the flow and large data sets that do not have to fit into memory. It's also the first streak surface method to explicitly treat shear flow.

3 Computational Model

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

1. Seed a curve using an interactive rake.
2. Update the streak surface by integrating every mesh vertex.
3. Refine the surface according to local deformation, by inserting/removing vertices/quads and updating the mesh connectivity.
4. Update the sampling rate of the vector field, by inserting and/or removing mesh vertices and joining them together using a quad-based topology.
5. Test boundary conditions such as object boundary intersection and zero velocity.
6. Render the surface.
7. This process iterates until the surface exits the space-time domain. At which point integration is terminated and the final surface is rendered.

Local operations are performed on quads and their neighbors which maintain sufficient sampling of the vector field. Vertex insertion is introduced when the Euclidean distance of a quad's edge is too long. Conversely a vertex is removed if neighboring points are too

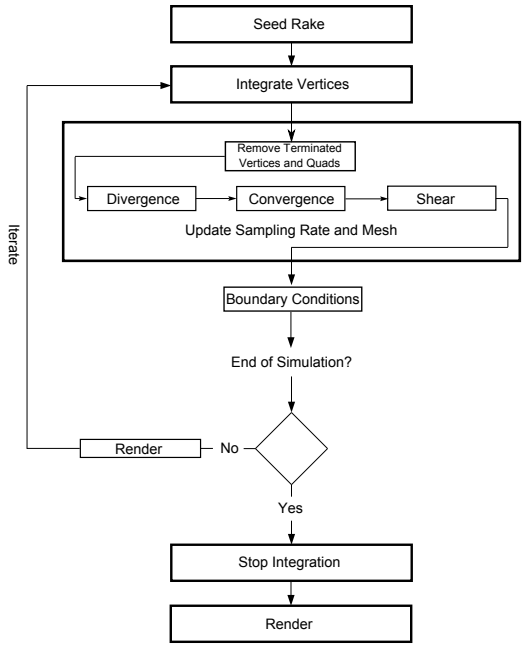

Figure 2: Algorithm flow chart.

close. Hence connectivity is then established ensuring all quad primitives are generally regular and that the surface is smooth. What follows is a description of each stage of our algorithm along with a description of the technical challenges and how they are addressed.

3.1 Streamlines, Pathlines, Streaklines, and Timelines

Trajectories of a vector field are streamlines. They are solutions to:

$$\frac{d\mathbf{x}}{ds} = \mathbf{v} \text{ or } \frac{dx_i}{ds} = \mathbf{v}_i(x_1, x_2, x_3, t) \quad (1)$$

Where s is a streamline parameter. Integral equation (1) results in streamlines *at the instant t*. Intuitively streamlines correspond to the path a massless particle traverses in steady flow. If we vary the temporal parameter, t, then we obtain *pathlines*: the path a single particle travels in unsteady flow.

The term streakline is used to denote the curve traced by a massless substance that is injected into the flow at a fixed point continuously over time. At time t, a streakline passing through a fixed point $\mathbf{y}$, defines a curve from $\mathbf{y}$ to $\mathbf{x}(\mathbf{y},t)$. The position of a particle coincides with this curve if it passes through $\mathbf{y}$ between time ϕ and t. The equation of a streakline at time t can be given by:

$$\mathbf{x} = \mathbf{x}[\zeta(\mathbf{y}, s), t] \quad (2)$$

where ζ is the initial coordinates of the point and where the parameter s lies in the interval $\phi \leq s \leq t$. Intuitively a streakline is the line joining all vertices passing through a position $\mathbf{x}$ at successive times. Dye injection/tracing is a very common method in laboratory flow visualization and streaklines allow for a direct comparison to this technique. A *timeline* is a line joining a series of vertices all released into the flow at the same time. Note that for steady vector fields, streamlines, pathlines, and streaklines are the same. We can view the streak surface mesh as a connected graph $\boldsymbol{G} = (V, E)$ where $V = \bigcup_i \boldsymbol{V}_i$, $\boldsymbol{V} \in \{\boldsymbol{V}_{NW}, \boldsymbol{V}_{NE}, \boldsymbol{V}_{SE}, \boldsymbol{V}_{SW}\}$ and $E \in \{E_N, E_E, E_S, E_W\}$. A quad is a sub-graph, $\boldsymbol{Q} = (\boldsymbol{V}_{NW}, \boldsymbol{V}_{NE}, \boldsymbol{V}_{SE}, \boldsymbol{V}_{SW}, E_N, E_E, E_S, E_W)$

3.2 Interactive Streak Surface Seeding and Advancement

We use an interactive seeding curve that allows the user to generate a variety of streak surfaces at run time and explore the whole domain. The user interactively controls the position and orientation of the seeding curve as well as its length and the number of streaklines

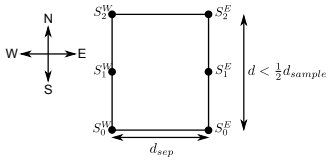

Figure 3: Line segments joining points along streaklines can be used to form quads. Edge lengths are $\leq \frac{1}{2}d_{sample}$. North is the direction pointing downstream from the seeding curve.

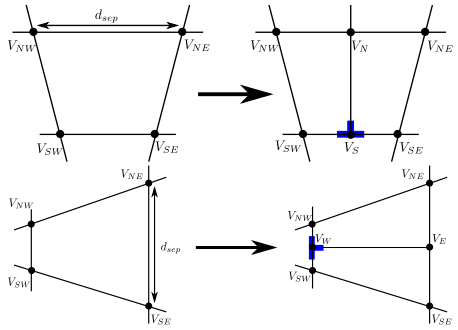

Figure 4: Our algorithm examines each quad edge. When $|V_{NE} - V_{NW}| > d_{sep}$ a quad is sub-divided. The same holds for when any quad edge length exceeds d_{sep}.

emanating from it. The default separating distance, d_{sep}, between streakline seeds is $\frac{1}{2}d_{sample}$ i.e., half the distance between neighboring data sample points on the underlying grid. This conforms to the Nyquist limit, namely, the sampling frequency must be (at least) twice that of the underlying data frequency for accurate reconstruction. This formulation is advantageous in that it can automatically adapt to changes in the data resolution i.e., adaptive resolution sampling. The seeding distance between points may be adjusted by the user if a more dense sampling is required or if faster performance is a priority.

As streakline points are integrated, we compute their distance from the seeding rake. As soon as a point, S_i, exceeds a distance of $\frac{1}{2}d_{sample}$ from the seeding rake, points S_i^W and S_i^E are joined to form a quad (see Figure 3). This allows us to adhere to the Nyquist limit by setting the advancement length to be half (or less) of the distance between the sample points of the underlying grid. This simple scheme allows the connectivity of the quad mesh elements to be composed of the corresponding points between adjacent streaklines, i.e., the points S_i^W, S_i^E, S_{i+1}^E and S_{i+1}^W, where i denotes the i^{th} point on the streakline (see Figure 3). We use the convention that north is the direction pointing downstream from the seeding curve.

As the streaklines elongate, their shape reflects the underlying characteristics of the unsteady flow. During their evolution, they encounter areas of shear, divergent and convergent flows. These areas require special handling.

3.3 Divergence

Divergence is a common characteristic in flow phenomena. When visualizing an area of divergent flow with a surface technique, the surface expands. Divergence in this case is defined when distance between the underlying flow lines used to construct the surface increases. By flow lines we mean timelines and streaklines. Streak surfaces present unique challenges for cases of divergent flow. Streak surfaces are dynamic in their entirety and therefore, divergence may occur anywhere within the surface. It is not restricted solely to the surface front.

Not only may the distance between adjacent streaklines increase,

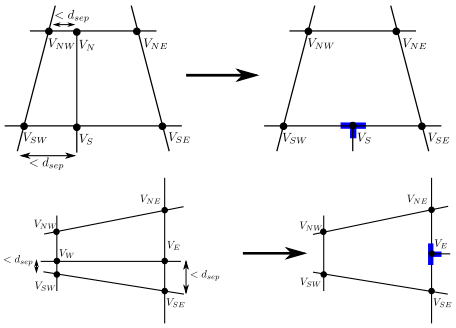

Figure 5: (Top) When $|V_{SW} - V_S| < d_{test}$ and $|V_{NW} - V_N| < d_{test}$, we collapse the two quads into a single one. (Bottom) When $|V_W - V_{SW}| < d_{test}$ and $|V_E - V_{SE}| < d_{test}$, we collapse the two quads into a single one. $d_{test} = d_{sep} - \varepsilon_{conv}$, where ε_{conv} is $\frac{d_{sep}}{2}$.

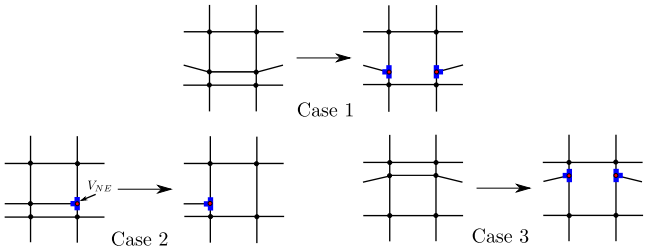

Figure 6: This figure shows changes in topology due to convergence. For each case the left hand side illustrates the state of the topology before the convergence operation has been applied. The right hand side shows the updated topology after the merge has taken place.

but the distance between neighboring points on the same streakline may increase as in Figure 4. In our framework, no distinction must be made between neighboring streakline or timelines due to the quad's symmetry. Our algorithm processes the quad mesh in a local quad-by-quad fashion. The edge lengths that define the quads are tested at each time-step to ensure that the streak surface maintains an optimal sampling frequency, i.e., to prevent under-sampling. To handle this, we simply divide the quad based on edge lengths longer than d_{sep}.

3.4 Convergence

Convergence occurs when the distance between flow lines decreases. As a consequence regions with a high density of points occur in the surface mesh. A high concentration of vertices in a local area may result in unnecessary oversampling. To prevent this we look at pairs of adjacent vertices and test the edge lengths we test for very thin quads, collapsing a pair of quads into a single one as in Figure 5.

Multiple ways of handling convergence are possible. Typically the removal of vertices is performed. Advancing front-based methods terminate an individual streamline/pathline and form a ribbon comprising of the neighbors of the terminated trace line. Garth et al. [Garth et al. 2008] propose not removing these vertices, claiming that the cost of handling convergent cases offsets the cost of integrating these additional points. There is also the possibility that the particular region of the surface may diverge again in future resulting in points being re-introduced, this may be avoided, or at the very least alleviated, if convergent points are not removed.

When converging the quads together, careful attention must be paid to the changes in mesh topology. The special cases are shown in Figure 6. The left-hand side of each case shows the mesh topology before mesh convergence is applied while the right-hand side shows the converged quads. In the implementation we first test to see which topological case the quad is in before updating the topology. More implementation details are given in Section 4 and in the supplementary material.

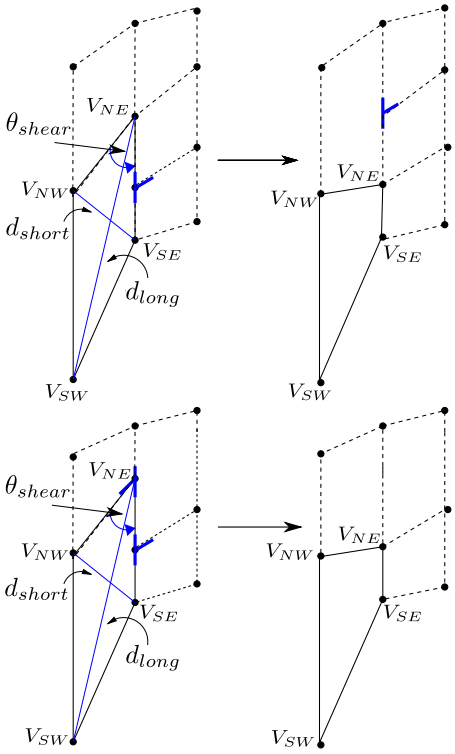

Figure 7: When a sheared quad is encountered a change in mesh topology results. In this example, the sheared quad has a T-Junction on its east edge, E_E. The Vertex, V_{NE}, is now updated to produce a more orthogonal quad. In the top row a T-junction is added to the north quad. $\theta_{shear} = 5°$

3.5 Shear Flow

Shear in the flow field presents difficult challenges when constructing a surface representation. Shear flow is problematic due to warping of the quad primitives. We define a pair of simple tests to determine the deformation of the quad. If the test indicates the quad is malformed we then alter the local topology of the mesh to produce a more regular mesh while maintaining it's accuracy. For the first test we compute the diagonals of the quad. The ratio $\frac{d_{short}}{d_{long}}$ can be used to quantify the amount of local shear. A perfect quad has a ratio of 1:1. If $\frac{d_{short}}{d_{long}} > \varepsilon_{shear}$ we then move onto the second test. The second test checks the subtending angle V_{NE}. If this angle is below the threshold θ_{shear} we then consider the quad as malformed and update the mesh connectivity ($\varepsilon_{shear} = 0.3$ and $\theta_{shear} = 5°$).

Figures 7 and 8 show how we handle shear flow. In Figure 7, the simple case, a T-junction is located on the edge E_E of the sheared quad (left column). When $\frac{d_{short}}{d_{long}} < \varepsilon_{shear}$ and $\theta_{shear} < \varepsilon_{\theta-shear}$ we re-connect the north edge, E_N, to the T-junction on E_E (right column). In the top row a new T-junction is introduced. In the bottom row, two T-junctions snap together.

In the second case, Figure 8, the left column shows the sheared quad before any update is made to the mesh topology while the right column depicts the updated topology. If $\frac{d_{short}}{d_{long}} < \varepsilon_{shear}$ and $\theta_{shear} < \varepsilon_{\theta-shear}$, we re-connect the sheared quads north-east vertex to the eastern neighbor's southwest corner. This forms an intermediate triangle. The triangle is then sub-divided into three quads – a method inspired by Alliez et al [Alliez et al. 2003]. In the top row, a new T-junction is created. In the bottom row, a T-junction is removed. The north direction of the new quad is depicted in Figure 8. This approach has the advantage that the original streakline points are re-used to handle the shear. Vertices are not moved, the

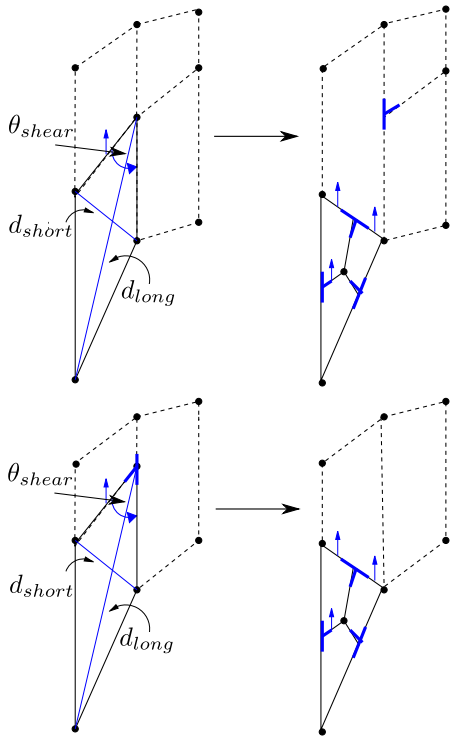

Figure 8: When a sheared quad is encountered a change in mesh topology results. In this example, the sheared quad has no T-junction on the east edge. The updated topology forms an intermediate triangle in the mesh. In order to maintain a quad-based topology we decompose the triangle into three quads using a method adapted from Alliez et al. [Alliez et al. 2003].

changes lie in the mesh topology.

There are also two possible cases when we subdivide the intermediate triangle. For each edge of the original quad we test to see whether there is a T-junction present. If there is no T-junction we insert a new vertex on that edge and interpolate it's position at the mid-point of the two edge vertices, this case is depicted in terms of the west neighbour of the original sheared quad in the top row of Figure 9.

3.6 Surface Discontinuities

It should be noted that handling divergence and convergence as in Figures 4 and 5 can cause T-junctions – a consequence of using quad meshes. T-junctions can be handled in a number of ways. Firstly, the vertices at the T-junction can be snapped to the edge causing the junction. For example, dragging V_S in Figure 5 (top) to edge E_S. Secondly, cracks appearing at T-junctions can simply be patched with triangles. The triangles are ignored during the integration and topology update phases. They can be re-applied afterwards in a separate pass. A third option is to only insert entire streaklines or timelines. This is recommended by Becker et al. [Becker et al. 1995]. The third option is costly in terms of performance. The first option involves moving points that have been previously calculated by the integration scheme and thus introduces error. This error may also accumulate over time.

In order to handle surface discontinuities we replace a quad that contains one or more T-junctions with a triangle-fan. The triangle fan uses one of the T-junctions as it's central vertex. It then proceeds clockwise adding vertices. We test each edge to see if it contains a T-junction, if one exists it's vertex is used in the triangle fan, if no T-junction exists we move onto the next quad vertex. Figure 10 illustrates this. If we don't consider T-Junctions when rendering,

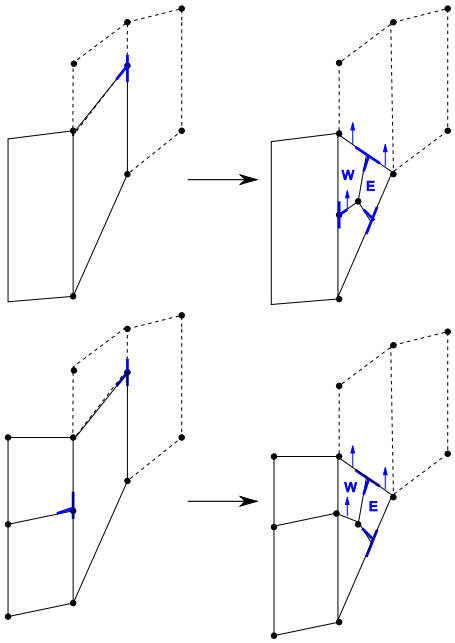

Figure 9: Left column: sheared quad, Right column: Updating the topology. When we subdivide the intermediate triangle, we test for existing T-junctions along its edges. If no T-junction is present, as in the top row, we insert a new vertex and interpolate it's position along the triangle edge. In the bottom row there is a T-junction present on the west edge of the triangle. In this case we simply use this vertex and it's position.

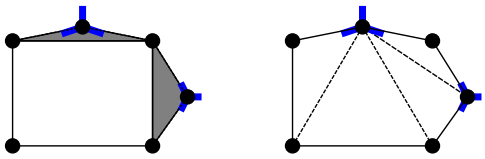

Figure 10: The image on the left shows a quad that contains T-junctions that do not lie on the quad's edge. The grey areas represent where cracks would be encountered in the mesh. The right image illustrates how our implementation handles this. A triangle fan is used, whose base is one of the T-Junction vertices. The triangle fan then proceeds in a clockwise manner connecting to the available vertices. This configuration allows for any combination of T-junctions.

discontinuities may appear in the mesh. This happens when the T-Junction vertex does not lie exactly on the quad edge to which it belongs. The triangle fan is ignored during the integration and used for rendering only.

4 Implementation

Our implementation involves three key objects: vertex, quad and T-junction objects. In this section we discuss these objects and how they relate to each other. We also provide the pseudo-code necessary for implementing the divergence, convergence and shear operations in a supplementary PDF file.

4.1 Mesh, Vertex, and Quad Objects

Mesh vertices contain three floats, each representing the x, y and z spatial components. They are stored in a central list. The ordering of the vertices may be arbitrary. The ability to compute vertices in any sequence allows us to add new vertices at the back of the list without re-ordering. A large number of vertices is typically

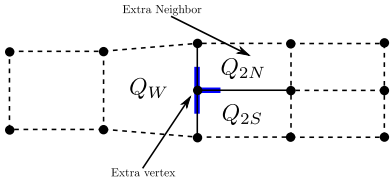

Figure 11: A T-Junction object. T-Junctions occur at a transition in mesh resolution caused by the divergence and convergence operations. The quad, Q_W stores the T-Junction object. The T-Junction object contains a pointer to the T-junction vertex, which can be used when this quad splits. This prevents duplicate vertices being inserted into the mesh. It also contains a pointer to the extra quad neighbor. The extra neighbor pointer is used when Q_{2N} splits, the newly inserted quad will point to Q_{2N}. Storing this explicitly prevents a search within the local region of the mesh, thus speeding up the computation.

removed from the surface due to convergence and exiting the spatio-temporal domain. The list structure aids in efficiency in this respect because we do not move all subsequent elements after the removal of a vertex.

Our quad objects are used to maintain the surface topology. They consist of four pointers to mesh vertices (which define the quad in the physical domain) and four pointers to neighboring quads, one for each direction: NORTH, EAST, SOUTH and WEST (where a NULL pointer indicates no neighbor). They also contain pointers to T-junction objects. A quad has a maximum of one junction and two quad neighbors per edge. If a pair of T-junctions is added on opposite edges of the quad, we simply split the quad. Quad objects are stored in a vector. Like the mesh vertex array, quads can be stored in any order. All divergence, convergence and shear operations and rendering are performed on a per-quad basis by simply iterating over the vector. The order of the operations is important. Quads are tested in the following order (1) divergence, (2) convergence and (3) shear (as in Fig 2). Also only a single operation is performed on a quad in a single pass through the surface in order to simplify implementation: either divergence, convergence, shear or no operation. A convergence or shear operation may sometimes result in a divergence operation in the next pass.

4.2 T-Junction Objects

T-junction objects are used to handle the transition in mesh resolution. A T-junction occurs when a single quad's edge is neighbored by two quads. This can occur in cases of convergence and divergence. A T-junction object contains a pointer to the extra neighbor of the quad and a pointer to the extra vertex, e.g. Q_W in Figure 11. When a quad that contains a junction splits we use the T-Junction vertex to ensure that the split quad shares the relevant vertices with it's neighbors. This also serves to prevent the unnecessary insertion of extra vertices. In the case where we add a T-junction object to a quad which already contains a T-junction on the opposite edge, we simply split the quad using the two T-junction points. T-Junctions are allocated dynamically when needed. They are removed when the quad is removed or when it splits.

4.3 Updating Mesh Topology

The supplementary PDF file provides a detailed discussion of the changes to topology that arise due to divergence, convergence and shear. We provide detailed diagrams and pseudo-code in order to facilitate implementation by others. Figure 12 depicts a subtle case where care must be taken when updating the mesh connectivity. This ensures that each quad is pointing to the correct neighbors.

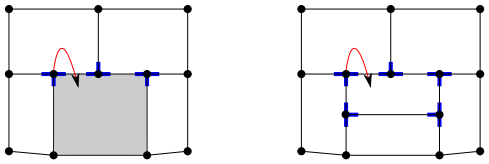

Figure 12: A subtle divergence case. We test whether the quad to the north is pointing to this quad or if it has a T-Junction that is pointing to this quad. We then update that pointer to point to the newly inserted quad. If this is not done the north quad will point to the incorrect quad.

5 Results

The reader is encouraged to view the accompanying video. In Figure **??**, a complex streak surface is constructed. This surface shows interesting flow structures that would be difficult to see if only streaklines were rendered. Surfaces also aid in identifying the transition of turbulence be showing stretching and folding. Our system constructs and renders this streak surface at approximately 3 frames per second (fps).

Figure 13 (top) depicts a complete streaksurface exhibiting a multitude of flow characteristics. This surface contains regions of divergence, convergence and shear as well as the splitting behavior when an object is encountered. This surface is computed and rendered at roughly 2 fps. Color is mapped to velocity magnitude in all our examples unless stated otherwise. Figure 13 (bottom) depicts another semi-transparent streak surface generated from the simulation of flow past a cuboid. The result demonstrates the tearing of the surface into two independent regions when an object boundary is encountered. Splitting is detected when the velocity magnitude of an internal streakline drops below a given threshold. When the surface tears, the separated wavefront are advanced independently. Figure 14 shows streak surfaces generated on a time-dependent tornado simulation. The tornado exhibits large regions of shear flow and is ideal to test the robustness of our shear-handling method. Our implementation renders this at 10 fps.

Large Data Sets Our system supports out of core memory management in order to handle large data sets. We adopt a method similar to Bürger et al [Bürger et al. 2007]. We store as many timesteps as possible in main memory. We then employ a sliding window. When performing the particle advection, we interpolate between a pair of timesteps. The management of timesteps is performed in a separate thread. The main thread is dedicated to the computation of the surface. This provides a benefit on multi-core systems as the blocking I/O call does not prevent the rendering and construction of the surface, while the relevant timesteps are loaded. This enables simultaneous streaming of data and the construction of the surface. Care must be taken to lock the portion of working memory in order to prevent incorrect values from being used in the surface computation.

Figure 15 depicts a streak sheet on the full resolution (500x500x100x48) Hurricane Isabel simulation. A streak sheet is created by simply terminating the insertion of new points at the seeding rake after a period of time. In this case the sheet was released so that it was captured by the eye of the hurricane. It then traces the hurricane's path. Figure 16 depicts a streak surface of an Ion Front Instability simulation. This is a high resolution turbulent data set with a resolution of 600x248x248 and consists of 200 timesteps.

Performance Table 1 shows performance timings for a sample streak surface. The table presents the performance timings for the

No. Vertices	RK2 Int.	Update Mesh: Divergence, Convergence, Shear	Total
10k	16	15	31
21.5k	47	15	62
50k	109	32	141
78.5k	171	32	203
100k	203	47	250
200k	406	93	499
500k	1030	1310	2340

Table 1: Performance times for our implementation given in milliseconds. We report both the time taken for the Runge Kutta 2 integration of the vertices and also the time taken to update the mesh topology after the integration phase. The results show that the integration process takes a large proportion of the computation time.

integration phase and the subsequent updating of the topology. The results show that the integration phase comprises a large proportion of the computational effort – typically 40-80% of the computation time. The mesh vertex advection is a largely parallel component, however our implementation performs this operation within a single thread. A multi-threaded version of this stage would greatly reduce the cost of this stage and consequently speed up the algorithm. However even in its current single-threaded state the algorithm generally still performs at interactive rates. While not as fast as a GPU-based version [Bürger et al. 2009] the mesh does not need to fit in graphics card memory, and thus handles larger streak surfaces. In principle, previous algorithms for unstructured data would run faster on structured data [Krishnan et al. 2009]. However, no implementation has been provided that demonstrates this. The bottom row in the table also shows a case of the updating of the mesh requiring more computational effort than the integration stage, due to a large number of divergent cases in the simulation data.

6 Conclusion

We present a novel interactive streak surface construction algorithm for the visualization of 3D unsteady flow. It combines the advantages of both speed and size with an efficient CPU-based, platform-independent implementation that places less restriction on memory and precision than a GPU-version. It is this quad-based approach from which the speed of the algorithm stems. It avoids expensive mesh re-triangulations. The local nature of the operations applied to the quad primitives enables the algorithm to be applied to large data sets (see accompanying video). The algorithm handles flow divergence, convergence, and shear. The algorithm also allows the surface to split when it meets object boundaries. The algorithm is demonstrated on a variety of data sets posing various challenges such as turbulence and large-scale simulations. A detailed CPU-based, platform-independent implementation is provided in the supplementary material to facilitate incorporation into visualization applications.

As future work we would like to treat the case of strongly deformed non-planar quads. Our model here does not treat them explicitly because we have not encountered this problem in our current simulation data sets.

References

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

BECKER, B. G., LANE, D. A., AND MAX, N. L. 1995. Unsteady Flow Volumes. In *Proceedings IEEE Visualization '95*, 329–335.

BÜRGER, K., SCHNEIDER, J., KONDRATIEVA, P., KRÜGER, J., AND WESTERMANN, R. 2007. Interactive Visual Exploration of Unsteady 3D Flows. In *Proc. EuroVis*, 251–258.

BÜRGER, K., FERSTL, F., THEISEL, H., AND WESTERMANN, R. 2009. Interactive Streak Surface Visualization on the GPU. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (November-December), 1259–1266.

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

GARTH, C., KRISHNAN, H., TRICOCHE, X., BOBACH, T., AND JOY, K. I. 2008. Generation of Accurate Integral Surfaces in Time-Dependent Vector Fields. *Proceedings of IEEE Visualization 2008* (Oct.).

HULTQUIST, J. P. M. 1992. Constructing Stream Surfaces in Steady 3D Vector Fields. In *Proceedings IEEE Visualization '92*, 171–178.

KRISHNAN, H., GARTH, C., AND JOY, K. I. 2009. Time and streak surfaces for flow visualization in large time-varying data sets. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (Oct.), 1267–1274.

LARAMEE, R. S., GARTH, C., SCHNEIDER, J., AND HAUSER, H. 2006. 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)*, Eurographics Association, 155–162,368.

MCLOUGHLIN, T., LARAMEE, R. S., PEIKERT, R., POST, F. H., AND CHEN, M. 2009. Over Two Decades of Integration-Based, Geometric Flow Visualization. In *Eurographics 2009: State-of-the-Art Reports*, 73–92.

MCLOUGHLIN, T., LARAMEE, R. S., AND ZHANG, E. 2009. Easy Integral Surfaces: A Fast, Quad-based Stream and Path Surface Algorithm. In *Proceedings Computer Graphics International 2009*, 67–76.

MCLOUGHLIN, T., LARAMEE, R. S., PEIKERT, R., POST, F. H., AND CHEN, M. 2010. Over Two Decades of Integration-Based, Geometric Flow Visualization. *Computer Graphics Forum*, forthcoming.

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

SCHEUERMANN, G., BOBACH, T., HAGEN, H., MAHROUS, K., HAMANN, B., JOY, K. I., AND KOLLMANN, W. 2001. A Tetrahedral-based Stream Surface Algorithm. In *Proceedings IEEE Visualization 2001*, 151–157.

SCHNEIDER, D., WIEBEL, A., AND SCHEUERMANN, G. 2009. Smooth Stream Surfaces of Fourth Order Precision. *Computer Graphics Forum 28*, 3, 871–878.

TONG, Y., ALLIEZ, P., COHEN-STEINER, D., AND DESBRUN, M. 2006. Designing Quadrangulations with Discrete Harmonic Forms. In *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 201–210.

VAN WIJK, J. J. 1993. Implicit Stream Surfaces. In *Proceedings of Visualization '93*, 245–252.

VON FUNCK, W., WEINKAUF, T., SAHNER, J., THEISEL, H., AND HEGE, H.-C. 2008. Smoke Surfaces: An Interactive Flow Visualization Technique Inspired by Real-World Flow Experiments. *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization 2008) 14*, 6 (November - December), 1396–1403.

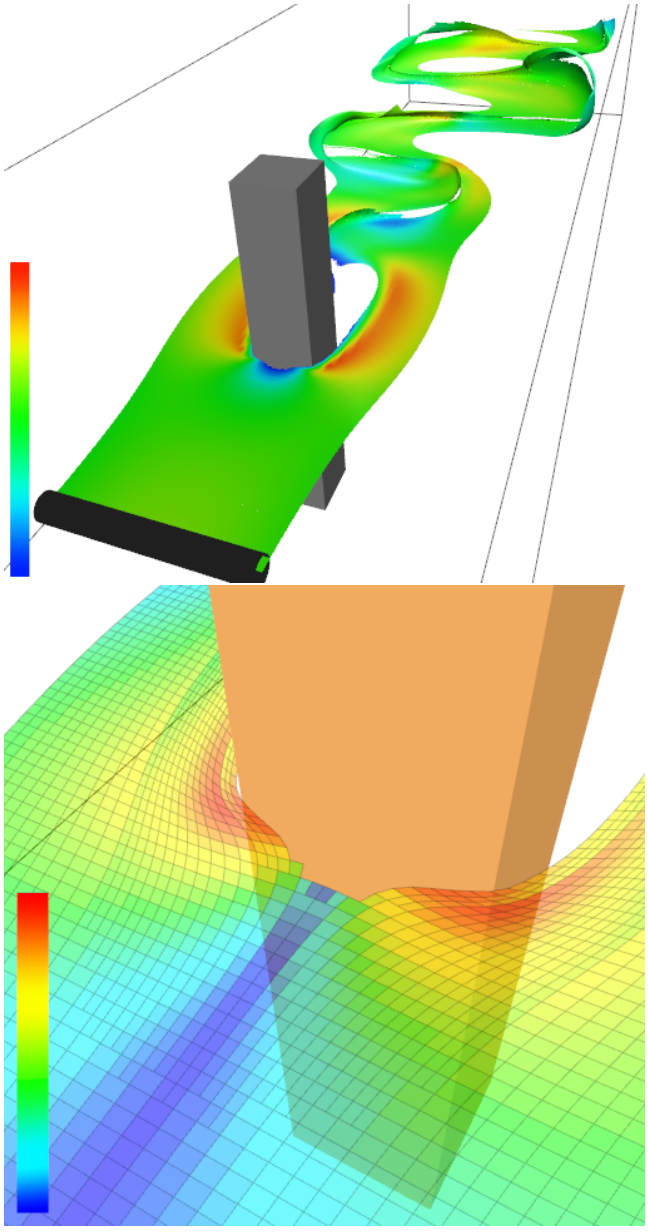

Figure 13: (Top) A streak surface of the simulation of flow past a square cylinder. The image shows a late stage into the simulation and shows the complete surface exhibiting divergence, convergence, shear and splitting behavior. (Bottom) When an object boundary is encountered the surface tears and moves around the boundary. This surface encounters a cuboid. As the surface splits the seperate portions are constructed independently of each other. Colour is mapped to the local deformation of the quad primitives (how much they deviate from being a regular quad.)

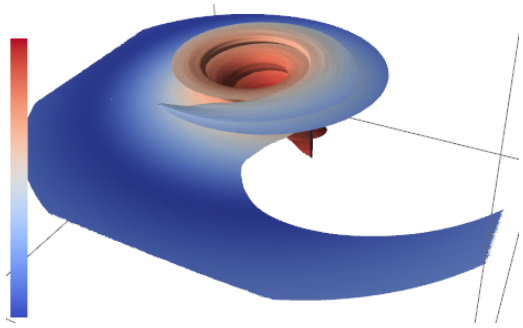

Figure 14: A streaksurface depicting a tornado simulation. The surface encounters large amounts of shear that accumulate as the simulation progresses.

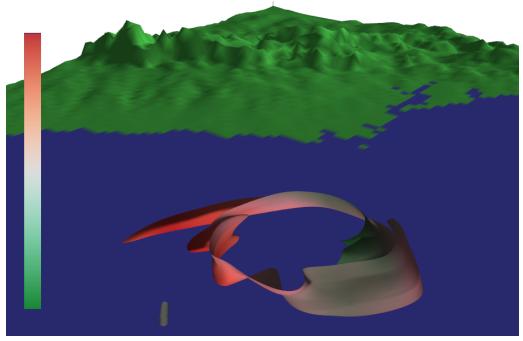

Figure 15: This image depicts the surface generated from the full resolution (500x500x100x48) of the Hurricane Isabel simulation. In this image we stop seeding the surface after a period of time and advect the sheet. Here the sheet is caught by the eye of the Hurricane.

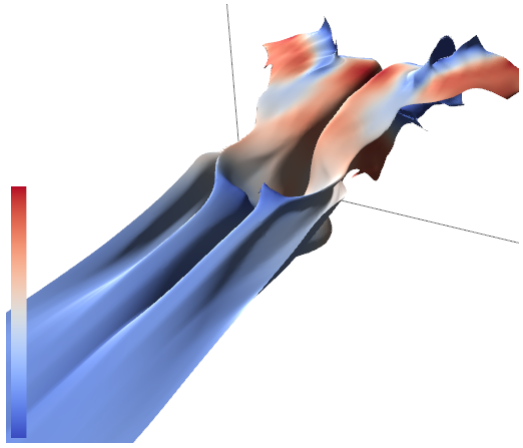

Figure 16: A streak surface on the Ionization Front Instability simulation (from the IEEE Vis'08 contest). This is a turbulent data set with 200 timesteps at a resolution of 600x248x248.