

A Level-Set Method for Flow Visualization

Rüdiger Westermann[†], Christopher Johnson[‡], and Thomas Ertl[‡]

[†]Scientific Computing and Visualization Group, University of Technology Aachen

[‡]Scientific Computing and Imaging Institute, University of Utah

[‡]Visualization and Interactive Systems Group, University of Stuttgart

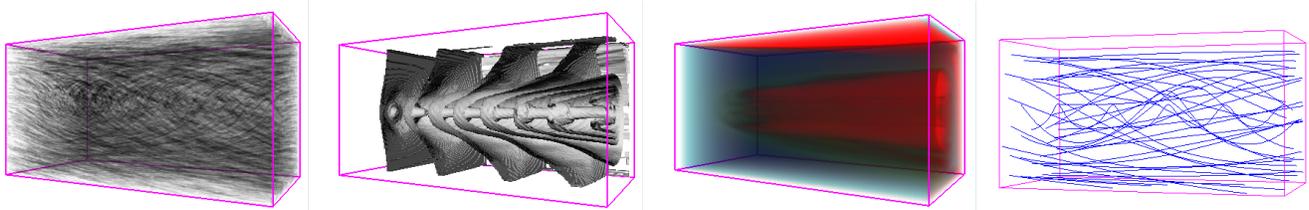


Figure 1: Dense flow fields are first converted into a scalar field, and then displayed and analyzed by means of level-sets in this field. An oracle that is based on the discrete curvature of level-sets allows for the automatic separation and extraction of homogeneous streams.

Abstract

In this paper we propose a technique for visualizing steady flow. Using this technique, we first convert the vector field data into a scalar level-set representation. We then analyze the dynamic behavior and subsequent distortion of level-sets and interactively monitor the evolving structures by means of texture-based surface rendering. Next, we combine geometrical and topological considerations to derive a multiscale representation and to implement a method for the automatic placement of a sparse set of graphical primitives depicting homogeneous streams in the fields. Using the resulting algorithms, we have built a visualization system that enables us to effectively display the flow direction and its dynamics even for dense 3D fields.

Keywords: Flow Visualization, Level-Sets, Feature Extraction, Multiscale Representation, Texture Mapping

1 Introduction and Related Work

Visualizing vector field data is challenging because no existing natural representation can visually convey large amounts of three-dimensional directional information. In fluid flow experiments, external materials such as dye, hydrogen bubbles, or heat energy are injected into the flow. The advection of these external materials can create stream lines, streak lines, or path lines to highlight the flow patterns. Analogues to these experimental techniques have been adopted by scientific visualization researchers. Numerical methods and three-dimensional computer graphics techniques have been used to produce graphical icons such as arrows, motion particles, stream lines, stream ribbons, and stream tubes that act as three-

dimensional depth cues. While these techniques are effective in revealing the flow fields' local features, the inherent two-dimensional display of the computer screen and its limited spatial resolution restrict the number of graphical icons that can be displayed at one time.

Additional techniques for flow field visualization include global imaging techniques. Crawfis and Max [5, 6] proposed direct volume rendering methods to create images of entire vector fields. Vector kernels and texture splats are used to construct three-dimensional scalar signals from the vector data. Van Wijk [28] proposed a Spot Noise method using stretched ellipses to create two-dimensional textures that can be mapped onto parametric surfaces. Max *et al.* [17] further utilized the spot noise method to visualize three-dimensional velocity fields near contour surfaces. Cabral and Leedom [3] presented a Line Integral Convolution (LIC) method, which makes use of a one-dimensional low pass filter to convolve an input texture along the principal curves of the vector field. Based on this idea, a number of related techniques have been proposed, which attempt to optimize the LIC method in terms of computational cost and image quality, to visualize flow over surfaces, and, most recently, to visualize 3D flow in a volume [23, 9, 1, 22, 13, 21].

These methods can successfully illustrate the global behavior of vector fields; however, it is difficult when using such methods to effectively control stream line density in a way that depicts both the direction structure of the flow *and* the flow magnitude. Furthermore, because of the tremendous information density they produce and their inherent occlusion effects, LIC methods have difficulty effectively visualizing 3D flow fields globally.

One approach to overcome these limitations is to interactively but manually modify the renderable representation in order to highlight the interesting structures [19]. Although visually pleasant results can be achieved by exploiting hardware-accelerated 3D texture mapping, in particular for large-scale vector fields it is difficult using this approach to detect the relevant structures without explicit knowledge concerning the underlying flow. Regardless of the inherent interactivity, this approach does not guarantee, in general,

that the characteristic flow features are found.

A different approach is to inspect the flow field in order to detect and analyze critical points [11]. In this approach, topological skeletons, which are defined by those stream lines starting at a critical point in the direction of the eigenvectors, are extracted and displayed. Although these techniques provide an effective tool to determine the topological equivalence of different flows, they sometimes do not yield an intuitive analysis of the principal streams and their direction.

Other techniques try to reduce the primitives used to depict the structure of the flow in such a way that the result still sufficiently represents the original data. While in [26] stream line placement in 2D flows is guided by visual attributes, in [14] evenly-spaced stream lines are generated based on a distance criterion but without explicit consideration of the flow topology. On the contrary, the main concern of the work presented in [10, 25] is to effectively simplify the underlying data without loss of relevant information. In general, however, these hierarchical techniques are local in that they usually consider only the vector field in the geometric neighborhood around each position, but do not take into account the global structure of the flow.

In this paper, we present a novel approach for the analysis and display of stationary vector field data, which includes effective techniques for the classification, segmentation and smoothing of flow fields. Rather than analyzing the flow field as such, we first convert it into a scalar field and then analyze the spatial and temporal evolution of level-sets or time surfaces in this field. In particular, we show how to obtain the flow data at ever coarser resolution by dispersing small disturbances across the time surfaces. In addition, we introduce two beneficial extensions of 3D texture-based iso-surface rendering, which allow for the simultaneous display of multiple, two-sided lighted surfaces and their dynamics using color table animations.

The goal of our approach is twofold: to obtain better visualization of the underlying structures by automatically placing a sparse set of graphical primitives depicting homogeneous streams in the flow *and* to generate a multiscale representation, which provides improved methods for particle tracing. In the latter case, the focus lies on developing an effective scheme that allows one to obtain accurate particle traces but with fewer total integration steps.

The remainder of this paper is organized as follows. First, we introduce the basic idea of converting the vector field data into a level-set representation and describe our extensions for interactive display of multiple level-sets via 3D textures. We then propose an explicit scheme for the smoothing of flow fields and demonstrate how to automatically select and place the graphical primitives depicting homogeneous streams. We conclude with a detailed discussion of results illustrating our approach applied to real data sets.

2 Flow Surfaces

In fluid dynamics, flow surface techniques have become important to the investigation of the dynamics of vector field data. A flow surface can be seen as a variation of path lines in non-stationary flows where several lines are joined to form a surface. A dense set of particles is released into the flow, and their subsequent positions, as well as the distortions of the so-defined surfaces, are monitored.

In computational flow visualization, techniques for simulating different kinds of flow surfaces have been developed in the past [12, 27, 4]. In its most general form, flow surfaces are simulated by placing an initial surface in the flow and then by successively moving all vertices defining the surface within constant intervals along the integral curves of the flow. The integral curves emanating at a given position are the solutions to the ordinary differential equation

$$\frac{d\mathbf{r}(t)}{dt} = \mathbf{v}(\mathbf{r}, t) \quad (1)$$

with initial boundary condition $\mathbf{r}(t_0) = \mathbf{r}_0$. Here, $\mathbf{r}(t)$ denotes the position of a particle at time t , and $\mathbf{v}(\mathbf{r}, t)$ represents the instantaneous velocity field. Notice in particular that this technique can be extended to the stationary case, in which a time-independent velocity field is considered and its integral curves are defined with respect to any other parameterization.

In general, flow surfaces can be placed everywhere in the flow; however, without loss of generality let us assume that particles are initially released at the inflow boundaries and at every source into all possible directions of the velocity at that source. Thus, the evolving surfaces formed by connecting particles at time t_n contain all positions within the domain that can be reached from a source in that time. In other words, any particle that is released from this surface and traverses its integral curve backward will reach a source or the boundary of the domain in that time. We will subsequently call these kinds of particles *anti-particles*, and this particular kind of flow surface the *time surface*.

A particular time surface in a flow is described by the implicit equation $T(x, y, z) = t_n$, where T is the time the anti-particle released at position (x, y, z) requires to reach a source or the boundary. Therefore, a time surface can be computed either by distorting the initial surface with respect to the flow field or by computing the scalar function T for all necessary positions and by fitting the surface using traditional techniques. The first approach has two major limitations: that it requires the generation and display of a large number of primitives, and that the generated surfaces are likely to become non-manifold and potentially self-intersect. In comparison, the second, implicit approach, which will be outlined in the following sections, has various advantages.

2.1 Level-set representation of time surfaces

We aim to construct a volumetric representation in which the time surfaces implicitly exist as level-sets. We define the level set of f at a particular value of time, t , as the set of all points (x, y, z) such that $f(x, y, z) = t$. Another name for this is the contour curve of f at level t . For an excellent introduction to level set methods and additional applications, see [20].

Therefore, for each grid point an anti-particle is released and its integral curve is traced until a source or the boundary of the domain is reached. This procedure is equivalent to the backward tracing approach proposed in [29] for the calculation of stream surface functions. We employ a fourth-order Runge-Kutta scheme with adaptive step-size control in order to find successive points along the curves. Prior to this procedure we determine critical points in the flow where the magnitude of the velocity vanishes. Thus, during run-time we detect anti-particles reaching a critical point, at which they must be stopped in order to avoid nonterminating traces. Additionally, a stopping criterion is employed for anti-particles moving on closed orbits, for which no valid time value can be assigned.

If the distance from point p_i to point p_{i+1} is d_i , then the time an anti-particle needs to travel from p_i to p_{i+1} is $t_i = d_i \cdot \frac{1}{|\mathbf{v}(p_i)|}$. Integrating the distances along the path yields the time the anti-particle needs to move from the grid point it was released from until it leaves the domain or reaches a critical point. A unique time is assigned to those grid points located in closed orbits. Thus, by storing all times at each grid point we have converted the vector field into a scalar field. This provides alternatives for display and analysis of the flow (see Figure 2).

2.2 Interactive display of level-sets

In order to effectively analyze a flow field by means of time surfaces, we need to interactively display the continuous evolution of these surfaces. As we have already pointed out, a geometric approach fails, in general; but once the flow field has been converted

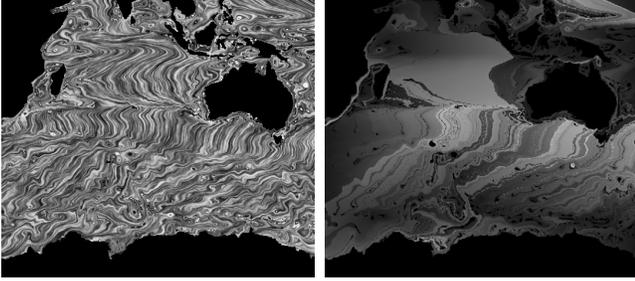


Figure 2: *In the left image, the ocean flow is depicted by means of LIC. On the right, the traveling times of anti-particles to the boundaries or sources are shown as scalar values at each grid point.*

into a level-set representation, we can exploit texture mapping hardware in order to show the intrinsic dynamic behavior. Though in the following we will shift emphasis to 3D flow fields, the basic approach can be utilized for the visualization of 2D vector fields as well.

Volume rendering via 3D texture maps has become a powerful tool to interactively display and thus analyze complex scalar data sets [2]. Interpreting volume rendering as the re-sampling of a discrete 3D texture map on cutting planes parallel to the viewing plane allows one to efficiently use hardware-supported texture interpolation to simulate the physics of light transport in semi-transparent media. The same principal technique can also be used to display lighted iso-surfaces [30] by re-sampling 3D gradient maps which store the pre-scaled gradients and the scalar data samples in a RGB and α texture, respectively.

The common procedure employed in ray tracing for iso-surface rendering, where the ray is traced until the first intersection with the surface is found, can now be simulated by means of OpenGL per-fragment operations. Combining alpha- and depth-tests during re-sampling guarantees that only those texture samples closest to the viewpoint and above/below a user-defined threshold are drawn into the frame buffer. Per-pixel diffuse lighting is accomplished by multiplying the RGB α components, which now store the gradient vector, with a color matrix [18] as available on SGI IR and Octane systems. This matrix has to be initialized properly to perform scaling, modelview rotation and the scalar product calculation with the light source direction vector. Key features of this extended approach are illustrated in Figure 3.

2.2.1 Multiple animated time surfaces

The method described in [30] allows only for the rendering of solid objects. Even more importantly, this method makes impossible the simultaneous display of surfaces corresponding to different iso-values or time-steps in the current application. Therefore we modified the algorithm by letting the α -values of texture samples be replaced by the contents of the texture color table before the per-fragment α -test is performed. Thus, by allowing texture samples to be drawn only if the α -value is greater than zero, we can render arbitrary time surfaces by simply windowing the appropriate α -range in the color table. This range can be arbitrarily scaled in order to change the thickness of the surfaces to be extracted.

In order to simulate the dynamics of the current flow, we have implemented cyclic shifts of the contents of the α -components in the texture color table. Now, we can show the direction in which the evolving structures proceed, as well as their speeds relative to each other.

2.2.2 Two sided lighting

We developed a two-pass approach that allows for the two-sided lighting of time surfaces. Therefore, we consecutively modulate pixel values with two different color matrices as shown below. Note that we show only that part of the matrices that is needed to perform diffuse lighting on the already scaled and rotated gradients. The first matrix, M_0 , includes light source direction (L_x, L_y, L_z) and its inverse to obtain the contributions from front- and back-sided lighting in the Red and Green pixel components, respectively. Note that negative values will be clamped to zero before they are drawn into the frame buffer. Thus, either the Red or the Green channel will be zero and can now be added in order to take over the appropriate values by multiplying the color components of each pixel with the matrix M_1 in the second pass.

$$M_0 : \begin{pmatrix} L_x & L_y & L_z & 0 \\ -L_x & -L_y & -L_z & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} M_1 : \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

In order to perform the color matrix multiplication and thus copy the frame buffer only once, we further optimized the outlined technique. The first row in matrix M_0 is duplicated into the 2nd and 3rd row. Now, after the color matrix has been applied we obtain color components in the range of $(-1, 1)$. Fortunately, prior to OpenGL clamping, these components can be scaled, biased, and finally mapped into the post color matrix lookup table. Thus, by issuing a scale and a bias of 0.5, and by initializing the color table with a two-sided ramp that ranges from one to zero in the first half and from zero to one in the second half, we arrive at the correct results. Although the accuracy of the results is limited by the width of the color table, in our tests no visual degradation of the image quality could be observed.

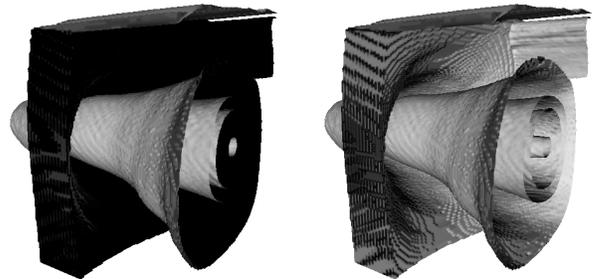


Figure 3: *These images illustrate the extensions we developed for 3D texture-based surface rendering. Multiple time surfaces with different thickness are rendered using one- and two-sided lighting.*

2.3 Stream boundaries

As we have claimed in the introduction, our approach should be effective in revealing homogeneous streams in the flow, which, in general, cannot be determined by just analyzing the vector data locally. Even if the vector data is locally homogeneous in terms of direction *and* speed, we will find regions where different streams proceed parallel to each other over a certain distance but will be separating again. As a solution to this problem, we have developed a local technique that takes into account global information by accumulating flow quantities along the integral curves.

Therefore, let us picture flux by treating it like the flow of a liquid, an imperfect analogy that is nonetheless useful for visualiza-

tion purposes. Let us consider the unsteady flux, and let us assume that the magnitude of the vector field, in some sense, gives us a measure of how much flux or matter is transported. Then, the time distribution we compute as proposed in Section 2.1 just indicates the net inward flux at a certain position along the line after a particular time. Since at each position we add the total incoming matter to the actual contribution, the difference between adjacent values on neighboring lines now indicates how much the accumulated matter along the line differs. Consequently, within homogeneous streams the distortions of level-sets corresponding to equal accumulated amounts are small in general, whereas they are high between different streams. We will subsequently call these kind of boundaries the *stream boundaries*.

Stream boundaries as introduced are defined by those positions where we have high variation in the changes of the flux at a certain time. Mathematically, this corresponds to positions where the Jacobian of the flux, $J = \frac{d\Phi}{dx}$, has high variation. Since the integral curves of the vector field, on the other hand, are the solutions to the differential equation $\frac{\partial\Phi(x,t)}{\partial t} = \mathbf{v}(\Phi(x,t))$ with initial boundary condition $\Phi(x_0, 0) = x_0$, J can be computed with respect to the following differential equation:

$$\frac{\partial}{\partial t} J = \frac{\partial}{\partial t} \left[\frac{d\Phi}{dx} \right] = \frac{d}{dx} \mathbf{v}(\Phi(t,x)) = J_{\mathbf{v}(\Phi(t,x))} \frac{d\Phi}{dx} = J_{\mathbf{v}(t)} \cdot J$$

with initial boundary condition $J(0) = I$, where I is the identity.

Note that the right hand side is time-dependent. In fact, the Jacobian of the vector field $J_{\mathbf{v}}$ along the whole path influences the Jacobian of the flow at a specified position x and time t . The Jacobian depends on the history along the path, thus leading to a non-trivial solution for J . In the next paragraph we will propose an analogue for computing J by curvature based measure.

If we change the inflow situation in terms of position, then the time until stream boundaries are formed changes as well. However, since we depict the variation of the flow with respect to the dynamic evolution along the streams and relative to each other, the feature lines will be featured independently of the inflow situation. Thus, by changing the inflow setting, we will also appropriately change the distribution of the scalar field T , while retaining boundaries between different streams.

2.4 Curvature based analysis of time surfaces

The study of time surfaces is of particular interest because they effectively visualize the geometric and topological modifications of their evolving structures. By helping us to discriminate among areas of flow showing different characteristics, these modifications should allow us to more accurately analyze the flow under consideration. As a consequence, we need to develop a measure for the variations of the flux as specified above that can be used to indicate the presence and the importance of stream boundaries.

One approach to detect and characterize ‘surface features’ in geometric modeling is to analyze the local curvature across the surface. Methods for the efficient calculation of the curvature can be found in many text books e.g. [8]. The use of this kind of information in surface fairing [24, 15, 7] is strongly related to our approach.

In the present scenario, the local curvature of the time surfaces tells us where distortions of these surfaces with respect to the influence of the flow field are most significant. Based on these observations, we want to derive a method that allows us to locally estimate the curvature of any iso-surface $T(x, y, z) = t_n$. Since we have already converted the flow field into the discrete level-set representation, we restrict our attention to the question of how to estimate the curvature of the trilinear interpolant within each cell of the underlying grid.

Therefore, let us assume that any time surface is defined as an elevation over the (x, y) plane and can thus be parameterized by the

equation $(x, y) \mapsto [x, y, \phi(x, y)]$. Here, we assume that a function $\phi(x, y)$ exists in the vicinity of $[x, y, z]$ such that

$$T(x, y, \phi(x, y)) = t_n.$$

Thus, we have an implicit description of the time surface at any regular point in the unit cube $[0, 1]^3$ that allows us to approximate any curvature measure within each cell. Typical measures for the curvature can be obtained from the first (g_{ij}) and second (h_{ij}) fundamental forms for the surface. With the standard notation T_x, T_y, \dots for the partial derivatives of T we obtain the total curvature

$$C = \frac{1}{4} (\kappa_1^2 + \kappa_2^2) = \frac{(T_x T_y T_{xy} + T_x T_z T_{xz} + T_y T_z T_{yz})^2}{(T_x^2 + T_y^2 + T_z^2)^3} + \frac{T_x^2 T_{yz}^2 + T_y^2 T_{xz}^2 + T_z^2 T_{xy}^2}{2(T_x^2 + T_y^2 + T_z^2)^2} - \frac{T_x T_y T_{yz} T_{xz} + T_x T_z T_{yz} T_{xy} + T_y T_z T_{xz} T_{xy}}{(T_x^2 + T_y^2 + T_z^2)^2} \quad (2)$$

of the surface $T(x, y, z) = t_n$ where all partial derivatives are evaluated at $[x, y, z]$.

In order to estimate the curvature within a certain cell, we evaluate equation 2 at randomly selected points in the interior of that cell. The maximum value is the curvature measure that will be used to analyze the time surfaces. High curvature within a cell indicates that grid points defining that cell belong to different streams. Thus, a stream boundary that isolates streams from each other passes through the cell.

As can be seen in Figure 4, the curvature plot naturally leads to the discrimination of separate streams that flow in different directions and/or with different speeds. In laminar streams where the distortions of the time surfaces are low, the curvature will be low as well. In the next section, we will demonstrate how to use this information to derive a multiscale representation for flow fields.

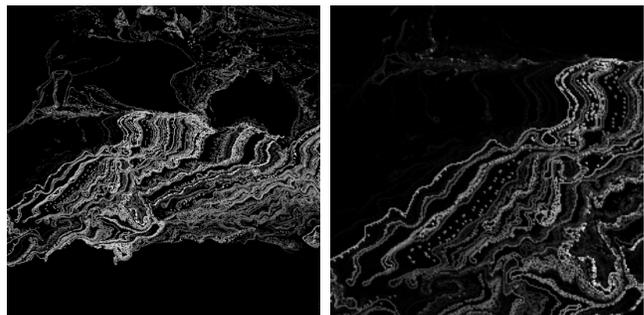


Figure 4: Two curvature plots of the time distribution for different inflow situations are shown. Note that small curvature values have been removed by thresholding.

3 Multiscale Flow Representation

The new curvature measure we have derived allows us to detect and separate streams that are homogeneous in terms of *direction* and *speed*. The feature lines we extract by means of this measure depict the boundaries between separable streams; within these streams the topology of the time surfaces is preserved over time and the main shape and length of stream lines is similar.

As with discrete fairing of meshes, where the geometry is smoothed with respect to the local curvature of the mesh, our goal

is to develop a technique that allows us to successively smooth the flow field with respect to the curvature of the time surfaces. Streams that have been separated should not be merged, whereas small deviations between the stream lines within them should be removed without significantly degrading their main shape.

3.1 Iterative smoothing scheme

Let us start by assuming that a local smoothing operator is available that can be employed to compute the incremental update to a flow vector with respect to its neighbors. Unfortunately, in contrast to the smoothing of meshes where the topology of the mesh tells us which of the neighbors have to be considered, this kind of topological information is not apparent in the flow field. Notice in particular that including all adjacent grid points into the local smoothing process results in the dispersion of disturbances across different streams as well as along the stream lines. However, neither effect is suitable, since both lead to undesirable smoothing orthogonal to the streams and equally undesirable distortion of the stream lines' main shape. In order to avoid these drawbacks, we have incorporated the local curvature into the smoothing process.

We start with a Cartesian grid and the initial distribution of the function values $T(x, y, z)$ at each grid point. We subsequently visit each voxel and locally reconstruct the time surface passing through that voxel by means of the marching cubes (MC) algorithm [16]. Note that we compute the discrete curvature not from the MC-surface but from the discrete time distribution T . Reconstruction is to be discontinued in cells where we have computed high curvature. Thus, we avoid including information from separate streams. At each vertex spanning the small piece of the time surface, we interpolate the velocity vectors V_j from the original flow field. The new velocity vector at the current grid point is obtained by inverse distance weighting with respect to the length of the edges l_j from this point to all V_j :

$$V_{new} = \sum_j w_j \cdot V_j \quad \text{with} \quad w_j = \frac{\frac{1}{l_j^2}}{\sum_i \frac{1}{l_i^2}}$$

Here, if $l_j < \epsilon$ then $w_j = 1$, and $w_{j \neq i} = 0$. We perform the same procedure to locally smooth the flow magnitudes. Finally, after processing all grid points, we end up with the smoothed flow field from which we compute the new values $T(x, y, z)$ to be used in the next iteration.

We conclude by shifting emphasis to a slightly different formulation of the proposed technique. For a certain time surface, a local smoothing operator as proposed in [24, 7] for the fairing of polygonal meshes could be employed. Given the surface corresponding to a specific time, the iterative solving scheme

$$V^{n+1} = (I - \lambda C)V^n$$

successively diffuses disturbances in the vector field across the time surface with respect to the damping factor λ and the discrete curvature C as derived in Section 2. In contrast to the fairing approach where positions of mesh vertices are updated, in our approach the vector field data V given on the vertices of the time surface is iteratively smoothed. In each iteration we derive a new vector field and the curvature has to be re-calculated.

3.2 Principal stream selection

Once we have constructed versions of the initial flow field in which the integral curves within separated streams are contained at ever coarser resolution, our goal is to place a sparse set of particles in the field and to show their stream lines. By taking into account the curvature values given at each grid cell, we intend to select the set

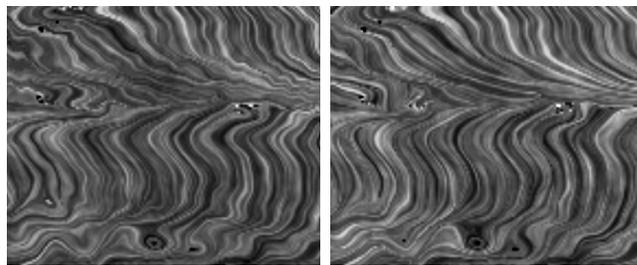


Figure 5: The left image shows a part of the original ocean data set. On the right the explicit smoothing scheme was applied using 9 iteration steps.

of particles in such a way that at least one particle trace is placed within each separated stream.

Therefore, we propose a technique that accounts for the topology of the time surfaces in the flow. We start by generating a binary data set C in the following way:

$$C[x_i, y_i, z_i] = \begin{cases} 1 & : \text{curvature}[x_i, y_i, z_i] \geq \epsilon \\ 0 & : \text{otherwise} \end{cases}$$

Now we randomly select positions in the flow field and recursively check whether there are cells $C[x_j, y_j, z_j] = 1$ in a certain region around that position. If so, we select a new position. Otherwise, the entire stream line passing through that position is traced and the binary field C is updated as follows:

$$C[x_j, y_j, z_j] = \begin{cases} 1 & : p \text{ hits } C[x_j, y_j, z_j] \\ C[x_j, y_j, z_j] & : \text{otherwise} \end{cases}$$

This procedure is repeated with as many particles as desired. The size of the region in which we check for cells that have already been set determines how close to the stream boundaries and to each other particle traces are placed (see Figure 6). As with the technique proposed in [14], we can now arbitrarily select the information density of the visualization.

In order to animate the dynamics along the streams, each extracted trace line is also stored as a set of line segments and the time value $T(\tilde{x}, \tilde{y}, \tilde{z})$ is stored for each vertex $(\tilde{x}, \tilde{y}, \tilde{z})$ that is included in the line. These values are then issued as *ID texture coordinates*, which allows the depiction of the relative speed along the lines by color table lookup as outlined in Section 2.

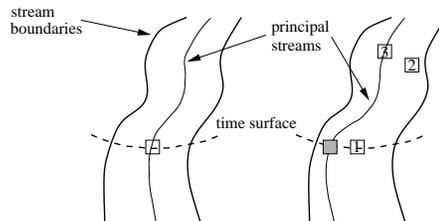


Figure 6: Principal streams are displayed for those cells that are far enough from the stream boundaries (left). On the right, cell 1 is not going to be selected because the time surface passing through it intersects the previously extracted principal stream passing through cell 3. Cell 2 is too close to a stream boundary and will be discarded as well.

As a consequence of the distance criterion, no lines will be placed in streams that are too thin or for which no particle has

been randomly selected. These streams can be easily detected by inspecting the set of cells in C that have not been set. By following the time surface passing through such a cell until a stream boundary is reached, we can check whether the surface intersects with a particle line ($C = 1$). If so, then a particle trace has already been selected and we proceed to the next cell. Otherwise, we either select an arbitrary cell the surface is passing through as the new seed point for a stream line, or we try to find the cell in the current stream that has a distance as equal as possible to all stream boundaries. This is done by shrinking the set of cells from the stream boundaries until only one cell remains. This will be the starting position of a new stream line. Shrinkage is similar to a morphological erosion, where we iteratively remove cells adjacent to stream boundaries, thereby narrowing the boundaries simultaneously and selecting a seed point in the middle of a stream (see Figure 7).

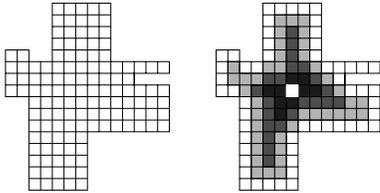


Figure 7: This illustration shows the narrowing of the discretized stream surface in order to find a seed cell in the ‘center’ of the surface. The procedure is stopped when we are left with one or multiple isolated cells.

Figure 8 shows the principal streams within the ocean fluid flow extracted with the proposed method and scan-converted into a 2D texture. Note that in high-turbulence regions where the curvature oscillates very irregularly, no principal streams are found because we do not consider positions that are completely surrounded by other high curvature cells due to the selected distance criterion. However, by changing our selection criterion appropriately we could use the same procedure to determine turbulent regions explicitly.

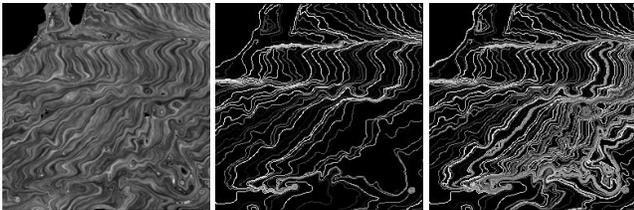


Figure 8: The leftmost image shows a part of the original data set. In the images on the right the principal streams are shown before and after thin or not yet found streams have been extracted.

4 Results and Analysis

In this section we discuss further results and analyze the main modules and features of our system. All tests were run on an SGI IR equipped with one R12000, 300 MHz processor, 64 MB texture memory and 256 MB main memory. Although our tests were restricted to Cartesian grids, we should mention here that other types of grids can be processed with only slight modifications. In particular, the particle tracer has to be modified appropriately, but above all, an algorithm is required that allows for the sampling of the vector data and for the curvature estimation. For curvilinear grids, for

example, for which we know how to compute partial derivatives in order to evaluate equation 2, our approach can be applied straightforwardly. For unstructured grids, however, we reconstruct the time surfaces locally and compute the discrete curvature on the triangle mesh. In any case, 3D textures have to be abandoned for rendering.

The most time consuming element of the presented approach is the computation of the time distribution T , which is accomplished by tracing the integral curves back in time until a source or the inflow boundaries are reached. The actual system implemented the Runge-Kutta scheme in a straightforward way without taking advantage of coherence in the data. The naive approach, in which every particle trace is computed from scratch, takes roughly 28 minutes for the 256^3 flow data set shown in Figure 10.

Once the scalar time field T has been generated, the time surfaces, as well as their dynamics, can be displayed interactively via 3D textures and color table animation. For example, on the used architecture the flow field mentioned before, including multiple time surfaces and color table animation, can be rendered with approximately 12 fps onto a 512×512 viewport using $256 \cdot \sqrt{3.0}$ slices.

The proposed multiscale representation enables us to remove noise from flow fields and to generate copies of the original flow at ever coarser resolution, as it is shown in Figure 5. One can easily recognize that the main shape of the stream lines is retained as high frequency oscillations are successively removed. Consequently, for an adaptive multi-step integration scheme that explicitly attempts to use fewer integration steps with decreasing size of details, the computational cost will be considerably decreased. In this way, unnecessary computations and invalid stream lines, which might occur when boundaries between separate streams are crossed, can be avoided effectively. In this respect, however, the criterion which lets us allow to separate streams from each other plays an important role.

As we have pointed out, our technique is intended to extract stream boundaries based on the proposed discrete curvature criterion. As a matter of fact, the classification of stream boundaries and consequently the smoothing process strongly relies on the curvature threshold we select as the importance measure. The specification of a proper error tolerance raises the same intrinsic problem as in other areas where techniques attempt to discriminate noise from features. On the other hand, although the discrete curvature is locally investigated, it gives a global measure because a point on the time surface carries information along the entire stream up to the current position. Consequently, noise along the stream lines will be increasingly removed due to integration, while we expect turbulence to introduce high frequency oscillations. This is due to the fact that flow direction will be changed significantly, thus altering the stream lines’ main shape.

Once the stream boundaries have been extracted by means of a curvature measure, the placement of the principal streams can be accomplished straightforwardly. Even without this step, when we initially place particle streams in the flow, it is guaranteed that all separated streams will be found employing the erosion like extraction of suitable seed points (see Figure 12). However, by just placing stream lines with a static distance to each other all streams broader than this distance can be extracted.

5 Conclusion

In this work we have emphasized a general approach for the visualization of flow fields by means of the dynamics of time surfaces. The major contribution here is to consider level-sets within this field as the fundamental structures showing the dynamics of the flow. The evolution of these level-sets in space and time is analyzed in terms of their local curvature, which enables us to separate homogeneous streams from each other.

We have developed two beneficial extensions for 3D texture-based iso-surface rendering allowing for the interactive and simultaneous display of multiple, two-sided lighted time surfaces and their evolution over time using color table animations.

We introduced an explicit scheme to effectively smooth flow fields. In particular, we have shown how to obtain stream lines at ever coarser resolution by dispersing small disturbances across the time surfaces at the same time retaining the integral curves' main shapes. Finally, this method has been extended for the automatic placement of principal stream lines in multi-dimensional flow fields. Thus, with a sparse set of lines, we are still able to indicate the relevant features in the data. However, our approach does have some drawbacks:

- Our approach is expensive in terms of numerical operations and storage for large-scale 3D flow fields. This is due to the voxel-wise grid traversal and the memorization of intermediate results.
- Our method relies on a heuristic curvature-based criterion. A more accurate investigation of the thresholding to be applied in order to separate streams from each other needs to be done.
- In particular, our method fails if the data is highly turbulent by nature, such that no regular stream boundaries can be detected. However, in this case no homogeneous streams exists - an important characteristic our approach is able to indicate.

Nevertheless, we are convinced that the ideas presented here will be influential for future developments:

- We have derived a multiscale representation for flow fields. This can lead to a multiresolution framework for flow, where only principal streams on a lower resolution level and the difference information needed for the next finer levels is included.
- We have demonstrated that flow field direction and speed can be visualized very effectively via texture-based rendering and by automatic extraction and placement of the principal streams. This enables us to appropriately visualize dense 3D flow fields.
- The multiscale representation might result in even more efficient integration schemes for particle tracing. By employing the multiscale nature of the analyzed flows, as well as their geometric and topologic structure, traditional integration schemes can be extended in order to control the step size adaptively and thus to improve their efficiency.

6 Acknowledgments

The authors would like to thank G. Greiner for valuable discussions and help. We also thank C. Reszk-Salama for the generous use of the LIC reference volume shown on the top of the color plate. We would like to thank NASA for providing the ocean data set. This work was supported, in part, by grants from NSF, NIH, and DOE.

References

- [1] H. Batke, D. Stalling, and H.-C. Hege. Fast line integral convolution for arbitrary surfaces in 3d. In *Visualization and Mathematics*, pages 181–195. Springer, 1997.
- [2] B. Cabral, N. Cam, and J. Foran. Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. In *Proceedings ACM Symposium on Volume Visualization 94*, pages 91–98, 1994.
- [3] B. Cabral and C. Leedom. Imaging vector fields using line integral convolution. In *Computer Graphics (SIGGRAPH 93 Proceedings)*, pages 263–270, 1993.
- [4] W. Cai and P.-A. Heng. Principal stream surfaces. In *Proceedings IEEE Visualization 93*, pages 75–80, 1997.
- [5] R. Crawfis and N. Max. Direct volume visualization of three-dimensional vector fields. In *Proceedings ACM Workshop on Volume Visualization*, pages 55–60, 1992.
- [6] R. Crawfis and N. Max. Texture splats for 3d scalar and vector field visualization. In *Proceedings IEEE Visualization 93*, pages 261–265, 1993.
- [7] Desbrun, M. and Meyer, M. and Schröder, P. and Barr, A. Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow. In *Computer Graphics (SIGGRAPH 99 Proceedings)*, pages 317–324, 1999.
- [8] G. Farin. *Curves and Surfaces for CAGD, 3rd edition*. Academic Press, 1993.
- [9] L.K. Forssell and S.D. Cohen. Using line integral convolution for flow visualization: Curvilinear grids, variable-speed animation, and unsteady flows. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):133–141, 1995.
- [10] B. Heckel, G. Weber, B. Hamann, and K. Joy. Construction of vector field hierarchies. In *Proceedings IEEE Visualization 99*, pages 19–27, 1999.
- [11] L. Hesselink and T. Delmarcelle. *Scientific visualization - advances and challenges*, chapter Visualization of vector and tensor data sets, pages 367–390. Academic Press, 1994.
- [12] L. Hultquist. Constructing stream surfaces in steady 3d vector fields. In *Proceedings IEEE Visualization 92*, pages 171–177, 1992.
- [13] V. Interrante and C. Grosch. Strategies for effectively visualizing 3d flow with volume lic. In *Proceedings IEEE Visualization 97*, pages 421–425, 1997.
- [14] B. Jobard and W. Lefer. Creating evenly-spaced streamlines of arbitrary density. In *Proceedings EG-ViSC '97*, pages 102–107, 1997.
- [15] L. Kobbelt. Discrete fairing. In *IMA Conference on the Mathematics of Surfaces*, pages 101–131, 1997.
- [16] W.E. Lorensen and H.E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *Computer Graphics (SIGGRAPH 87 Proceedings)*, pages 163–169, 1987.
- [17] N. Max, R. Crawfis, and C. Grant. Visualizing 3d velocity fields near contour surfaces. In *Proceedings IEEE Visualization 94*, pages 248–255, 1994.
- [18] T. McReynolds. Tutorial on programming with opengl: Advanced rendering. In *SIGGRAPH 96*, 1996.
- [19] C. Rezk-Salama, P. Hastreiter, and T. Ertl. Interactive exploration of volume line integral convolution based on 3d-texture mapping. In *Proceedings IEEE Visualization 99*, pages 233–240, 1999.
- [20] J.A. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University, Cambridge, 1999.
- [21] H-W. Shen and D. Kao. Uflic: A line integral convolution algorithm for visualizing unsteady flows. In *Proceedings IEEE Visualization 97*, pages 317–323, 1997.
- [22] H.W. Shen, K.L. Ma, and C.R. Johnson. Global and local vector field visualization using enhanced line integral convolution. In *Proceedings ACM Symposium on Volume Visualization*, pages 63–70, 1996.
- [23] D. Stalling and H.-C. Hege. Fast and resolution independent line integral convolution. In *Computer Graphics (SIGGRAPH 95 Proceedings)*, pages 249–256, 1995.
- [24] G. Taubin. A signal processing approach to fair surface design. In *Computer Graphics (SIGGRAPH 95 Proceedings)*, pages 351–358, 1995.
- [25] A. Telea and J. Wijk. Simplified representation of vector fields. In *Proceedings IEEE Visualization 99*, pages 35–43, 1999.
- [26] G. Turk and D. Banks. Image-guided streamline placement. In *Computer Graphics (SIGGRAPH 96 Proceedings)*, pages 453–460, 1996.
- [27] J. van Wijk. Implicit Stream Surfaces. In *Proceedings IEEE Visualization 93*, pages 245–260, 1993.
- [28] J.J. van Wijk. Spot noise: Texture synthesis for data visualization. *Computer Graphics*, 25(4):309–318, 1991.
- [29] J.J. van Wijk. Implicit stream surfaces. In *Proceedings IEEE Visualization 93*, pages 245–253, 1993.
- [30] R. Westermann and T. Ertl. Efficiently using graphics hardware in volume rendering applications. In *Computer Graphics (SIGGRAPH 98 Proceedings)*, pages 291–294, 1998.

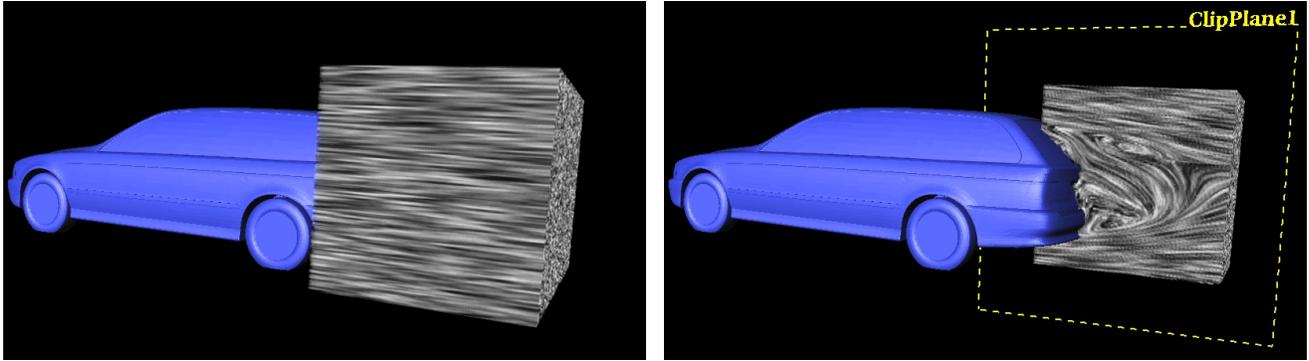


Figure 9: Both images show the LIC volume of the flow around the back of the car. Only by means of an additional, manually positioned clipping plane the relevant structures can be extracted (see color plate).

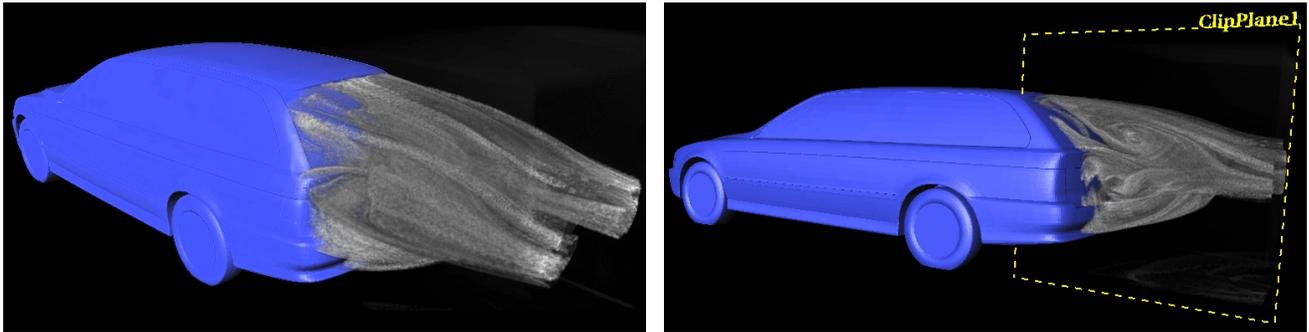


Figure 10: Both images show the curvature volume computed from the flow field around the back of the car. Dark grey indicates high curvature. The relevant structures can be clearly distinguished even without any manual modifications (see color plate).



Figure 11: First, we show stream lines automatically selected by our curvature based oracle in the ocean data set. Next, multiple two-sided lighted iso-surfaces in the engine data set and multiple time surfaces computed from the flow around the car are displayed. Note the fuzzy structures where we have turbulent flow (see color plate).

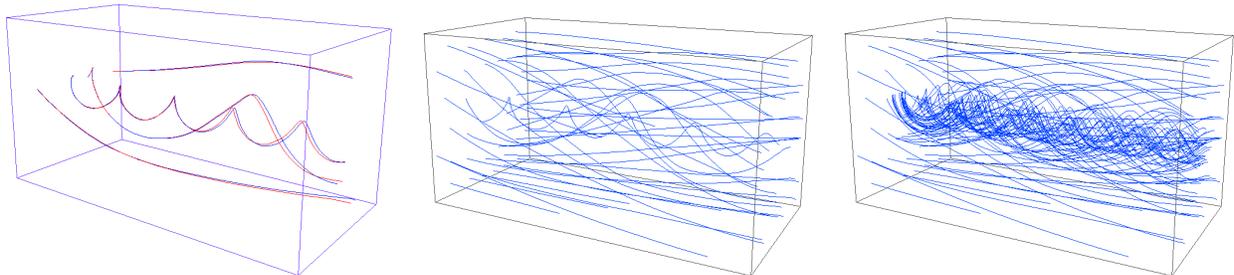


Figure 12: First, the deviation between stream lines before (colored blue) and after (colored red) smoothing the flow field is shown. Although the deviation is low, we need 30% less integration steps to generate the stream lines colored red. Next, we show the initially placed stream lines. Finally, additional stream lines are placed automatically in separable streams (see color plate).