Vision, Modeling, and Visualization (2012)
M. Goesele, T. Grosch, B. Preim, H. Theisel, and K. Toennies (Eds.)

Implicit Integral Surfaces

T. Stöter[1] and T. Weinkauf[1] and H.-P. Seidel[1] and H. Theisel[2]

[1]Max Planck Institute for Informatics, Saarbrücken, Germany [2]University of Magdeburg, Germany

Abstract

We present an implicit method for globally computing all four classic types of integral surfaces – stream, path, streak, and time surfaces – in 3D time-dependent vector fields. Our novel formulation is based on the representation of a time surface as implicit isosurface of a 3D scalar function advected by the flow field. The evolution of a time surface is then given as an isovolume in 4D space-time spanned by a series of advected scalar functions. Based on this, the other three integral surfaces are described as the intersection of two isovolumes derived from different scalar functions. Our method uses a dense flow integration to compute integral surfaces globally in the entire domain. This allows to change the seeding structure efficiently by simply defining new isovalues. We propose two rendering methods that exploit the implicit nature of our integral surfaces: 4D raycasting, and projection into a 3D volume. Furthermore, we present a marching cubes inspired surface extraction method to convert the implicit surface representation to an explicit triangle mesh. In contrast to previous approaches for implicit stream surfaces, our method allows for multiple voxel intersections, covers all regions of the flow field, and provides full control over the seeding line within the entire domain.

1. Introduction

Visualizing 3D flows using integral surfaces is common practice these days. There are four types of such surfaces, each of them suitable for a different scenario: stream and path surfaces trace out a family of particle trajectories started from a line in steady or unsteady flows, respectively. Streak and time surfaces mimic the behavior of smoke or dye in real-world experiments. Integral surfaces are an intuitive tool to reveal vortex structures, laminar flow, regions with convergent or divergent flow behavior, and many other flow features. Integral surfaces often prove advantageous over a set of integral lines, since surface shading, two-sided lighting, and hidden surface removal provide better perceptual cues.

At the same time, their computation is involved. Attesting to that fact are the large number of methods for computing integral surfaces. In one way or another, most of these methods build upon Hultquist's advancing front method [Hul92] for stream and path surfaces. Stalling [Sta98] as well as Peikert and Sadlo [PS09] developed extensions that work better in the proximity of critical points. Scheuermann et al. [SBM*01] exploited the linearity in tetrahedral meshes to create stream surfaces of high accuracy. Garth et al. [GKT*08] separated the integration and triangulation stages to achieve well-conditioned meshes. Recently, also streak and time surfaces have gained attention in the visualization community [vFWTS08, BFTW09, KGJ09].

All these methods compute integral surfaces *explicitly* as triangle or quad meshes. In other areas such as Geometric Modeling, both *explicit* and *implicit* representations of surfaces are well-established. In fact, in Geometric Modeling the relation between explicit and implicit representation is well-researched. It is known [BKP*10] that both representations have their own strengths and weaknesses, which are complementary in the sense that the strength of one representation is the weakness of the other, and vice versa. This has led to an intensive research in conversion algorithms between explicit and implicit representations. A disadvantage of explicit representations is that they come with a particular surface parametrization. It is challenging to ensure the quality of the parametrization during different surface operations. Contrary, implicit representations do not have this problem: they are parametrization-free by definition. Furthermore, implicit representations can usually live on simple regular grids. Moreover, an explicit representation describes one particular surface, while an implicit representation describes a whole family of surfaces at once.

Considering the points mentioned above, it does not come as a surprise that implicit methods have been developed for integral surfaces as well. The concept of *implicit stream surfaces* has been introduced by van Wijk [vW93] for steady 3D vector fields. The main idea is to advect a 2D scalar field from the domain boundary into the 3D domain: a stream line is traced backwards from each vertex of a uniform grid. If it reaches the boundary, the corresponding scalar value is assigned to the vertex. Figure 1 gives an illustration. This defines a 3D scalar field on a uniform grid whose isosurfaces are a family of stream surfaces of the vector field.

However, the concept of implicit stream surfaces has not been picked up by the Visualization community. We see the reason for this in three serious drawbacks of the approach in [vW93]:

DOI: 10.2312/PE/VMV/VMV12/127-134

- **Limited domain coverage**
  Stream lines may start and end inside the domain without ever reaching the boundary.This occurs due to sources, sinks, closed orbits, and vortex structures. The approach of [vW93] fails to create stream surfaces in these regions.
- **Limited voxel intersections**
  The isosurface of a trilinear scalar field intersects a voxel only a few times (usually only once). A stream surface, on the other hand, can intersect a voxel multiple times. This is especially the case in the vicinity of vortex structures, where the stream surface curls up and intersects a voxel repeatedly.
- **Limited control of the seeding line**
  The seeding line can only be chosen at the boundary, but not placed freely within the 3D domain.

Furthermore, the approach of [vW93] cannot easily be extended to path and streak surfaces, since they self-intersect in space, but an isosurface does not. In fact, the approach of [vW93] is limited to steady flows.

In this paper, we introduce a novel concept for implicit integral surfaces. In unsteady flows, it describes path, streak and time surfaces. It can also be applied to steady flows, where it yields implicit stream surfaces, but our algorithm solves all the problems mentioned above.

Our method is based on the advection of 3D scalar functions in the flow field (Section 2). This creates a 4D scalar field and we will show that its isosurfaces describe evolving time surfaces of the underlying flow. Based on that, we define implicit stream, path, and streak surfaces as intersections of two isovolumes in 4D (Section 3), for which we provide three novel visualization techniques (Section 4). We show applications of our method in Section 5.

To the best of our knowledge, we present the first approach that is able to handle path and streak surfaces implicitly. Furthermore, we are not aware of any other approach – implicit or explicit – that provides a single framework for all four integral surfaces at once.

Further Previous Implicit Approaches for Flow Visualization

Implicit stream surfaces are used in further approaches. Kenwright and Mallinson [KM92] compute stream lines as the intersection of different stream surfaces. Cai and Heng [CH97] define special implicit stream surfaces which aim to depict the topology of an irrotational flow.

It shall be noted that van Wijk [vW93] and Westermann et al. [WJE00] also provided a scheme to compute implicit time surfaces in steady flows. To this end, each vertex of the uniform grid is assigned the amount of integration time it takes from the boundary to the vertex. Again, these methods work only for steady flows.

For 3D unsteady flows, a number of methods have been developed for dye advection [Wei04, CKSW08]. These methods allow for path and streak *volumes* as well as time surfaces. A major difference to our approach is that we implicitly define an infinitely large family of integral surfaces, whereas dye advection methods aim at creating these structures for a small set of seeds.

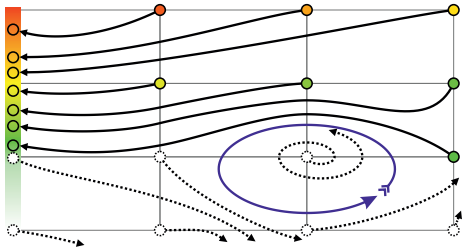


Figure 1: *Schematic sketch of van Wijk's [vW93] advection concept for implicit stream surfaces. A scalar value is assigned to a grid vertex based on where its stream line hits the boundary. Stream lines may hit different boundaries, which can be handled with some effort. But stream lines may also never reach the boundary due to e.g. critical points or closed orbits (blue ellipse). These regions cannot be handled by this approach.*

2. Background

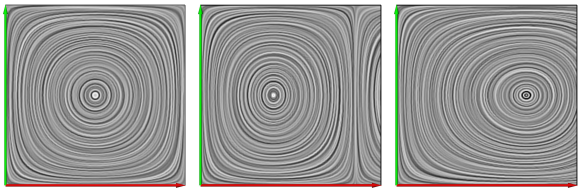
We consider a 3D time-dependent vector field $\mathbf{v}(\mathbf{x},t)$ over the spatial domain $D = [x_{min},x_{max}] \times [y_{min},y_{max}] \times [z_{min},z_{max}]$ and the temporal domain $T = [t_{min},t_{max}]$. We write derived $(3+1)$-dimensional variables with a bar like $\bar{\mathbf{p}}$, and derived $(3+2)$-dimensional variables with a double bar like $\bar{\bar{\mathbf{q}}}$. Locations in space-time are denoted as $\bar{\mathbf{x}} = (\mathbf{x},t)^T = (\mathbf{x},t)$. All vectors throughout the paper are column vectors, we often omit the explicit $()^T$ notation.

For explanations and illustrative purposes, we will often refer to the well-known time-dependent *Double Gyre* example from Shadden [Sha05] – originally a 2D unsteady flow, but here artificially blown up to a 3D unsteady flow to keep the illustrations simple and comprehensible. We consider it in the spatial domain $[0,1]^3$ and the temporal domain $[0,12]$.

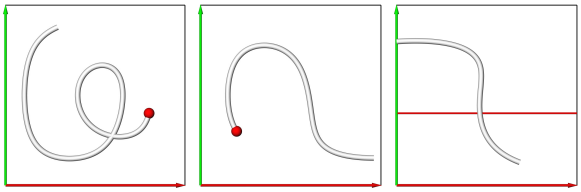
2.1. Integral Lines and Integral Surfaces

There are four classic types of integral lines for time-dependent vector fields: stream, path, streak and time lines. Figure 2 accompanies the following explanations. Stream and path lines are the trajectories of massless particles in steady or unsteady vector fields, respectively. A streak line is the locus of all particles passing through a specific location over time. They can easily be visualized in real-world flow experiments by means of dye injection. Note that stream, path and streak lines coincide in steady vector fields.

This is in contrast to time lines, which differ from the other ones in both steady and unsteady fields. In particular, they are not seeded at spatial point locations, but rather at a line. A time line is then the collection of all particles started from this line and integrated in the flow for some



(a) Stream lines at different time steps visualized using LIC.



(b) Path line seeded from red point. (c) Streak line seeded from red point. (d) Time line seeded from the red line.

Figure 2: *All four classic types of integral lines in the example vector field, shown in the xy-plane.*

time. An analogon in the real world is an infinitely flexible yarn or wire thrown into a river, which gets transported and deformed by the flow.

There is another distinction between stream/path lines on the one hand, and streak/time lines on the other hand: a stream/path line denotes the trajectory of a *single* particle *over* integration time, whereas a streak/time line gives the location of *several* particles at a *constant* time step.

In this paper, we are concerned with integral surfaces. They are simply families of the corresponding integral lines. Stream, path and streak surfaces are seeded from a line, while time surfaces are seeded from a surface.

We refer the interested reader for a more detailed analysis of integral lines and surfaces to Weinkauf et al. [WT10, WHT12].

2.2. Flow Maps and Scalar Field Advection

To describe implicit integral surfaces, we use the concept of flow maps. The flow map $\phi : D \times T \times \Upsilon \to D$ describes the spatial location of a particle seeded at $(\mathbf{x},t)$ and integrated over a time interval $\tau \in \Upsilon$, denoted as $\phi_t^\tau(\mathbf{x}) = \phi(\mathbf{x},t,\tau)$. In other words, ϕ maps the start point of a particle integration to its end point. Many recent flow analysis methods use this concept. For example, Finite Time Lyapunov Exponents (FTLE) [Hal01] or Streak Line Vector Fields [WT10] require the gradient of the flow map. Our work here requires only the flow map itself, not its gradient.

Furthermore, our method builds on the concept of advection, which describes the transport of a substance or quantity in a vector field – a 3D scalar function in our case. We use direct Lagrangian back tracking, which integrates backwards in time to find the original scalar value at the start of the advection process. In more formal terms: given are a continuous 3D scalar function $a_s : D \to \mathbb{R}$ and a certain time step t.

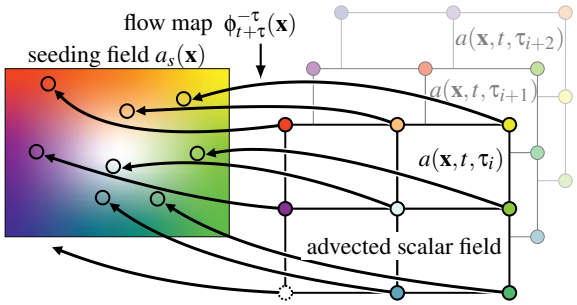


Figure 3: *Schematic sketch of our advection concept for implicit integral surfaces. A scalar value is assigned to each grid vertex based on the end point of a particle integration over a time interval τ. This is done efficiently by looking up the end point in the flow map ϕ. The advected scalar field depends on τ and the start time of the integration. It is defined everywhere except for particles that leave the domain.*

We refer to a_s as the *seeding field*. We want to advect $a_s(\mathbf{x})$ in the flow field over time from t to $t+\tau$. To achieve this, we integrate backwards from $t+\tau$ to t. This can be done efficiently by looking up the end point of the integration in the flow map. The *advected scalar field* $a(\mathbf{x},t,\tau)$ is then given as

$$a(\mathbf{x},t,\tau) = a_s\left(\phi_{t+\tau}^{-\tau}(\mathbf{x})\right) \qquad a : D \times T \times \Upsilon \to \mathbb{R}. \qquad (1)$$

Note that for $\tau = 0$, we have $a(\mathbf{x},t,0) = a_s(\mathbf{x})$, i.e., the advected scalar field is the seeding field before any integration took place. We will use one or more seeding fields later to implicitly define seeding lines and surfaces. Figure 3 illustrates our advection scheme.

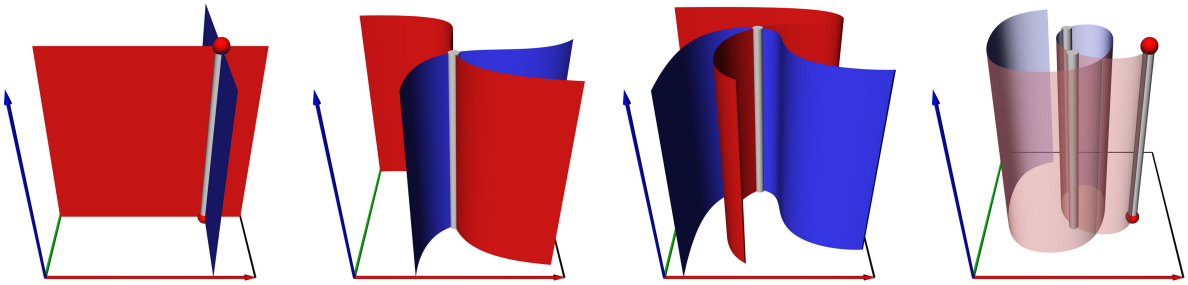
Implementation Details We compute the flow map as a dense integration (4th-order Runge-Kutta) of the vector field in a preprocessing step. It is sampled on an uniform grid with three spatial dimensions x,y,z, a time dimension t, and a dimension for the time interval τ. In most our applications, we choose a distinct time step such as t_{min} or t_{max}, which significantly reduces the amount of space required for the flow map. Furthermore, the flow map can be re-used for other visualization methods such as FTLE. The flow map is used directly to do the scalar advection. The seeding field can either be given as an analytic expression, or as a sampled scalar field on a grid.

3. Implicit Integral Surfaces

With our advection concept from (1) at hand, we can now define all four types of integral surfaces implicitly.

3.1. Implicit Time Surfaces

For time surfaces, we just need a single seeding field $a_s(\mathbf{x})$. Any isosurface $a_s(\mathbf{x}) = a_0$ can serve as an implicitly given seeding surface. Now we define a start time t_s for the advection and get the corresponding advected scalar field $a(\mathbf{x},t_s,\tau)$. This is a 4D scalar field where the τ-dimension



(a) Implicit definition of the seeding line as intersection of two isosurfaces of the seeding fields a_s, b_s.

(b) Both scalar fields are advected with the flow field. This advects the two isosurfaces and makes them time surfaces seeded from the surfaces in the first step. Note how their intersection line is advected as well.

(c) Collecting all intersection lines gives the path surface. *(Actual computations take place in 4D, see text.)*

Figure 4: *Implicit path surfaces as the intersection of two evolving time surfaces. Seeding fields: $a_s(\mathbf{x}) = x$ and $b_s(\mathbf{x}) = y$.*

describes the amount of time that has been passed since the time surface was seeded at t_s. The time surface of a certain τ_i is then given as the isosurface of the 3D scalar field

$$a(\mathbf{x}, t_s, \tau_i) = a_0. \tag{2}$$

We can observe the temporal evolution of the time surface by slicing through the τ-dimension: $\tau_0 \leq \tau_i \leq \tau_n$. Figure 5a shows that.

Note that we just defined an infinitely large family of time surfaces: changing the isovalue a_0 yields *instantly* a different time surface with its entire evolution. See Figure 5b.

Mathematically, we can describe the evolution of a time surface also as an *isovolume* in the 4D scalar field $a(\mathbf{x}, t_s, \tau)$. This point of view might be helpful for the following sections.

3.2. Implicit Path Surfaces

A path surface is seeded from a line. We define the seeding line implicitly using two seeding fields $a_s(\mathbf{x})$ and $b_s(\mathbf{x})$ together with two isovalues a_0, b_0. Then

$$[\, a_s(\mathbf{x}) = a_0 \;,\; b_s(\mathbf{x}) = b_0 \,] \tag{3}$$

gives a line structure as solution: it is the intersection of the two isosurfaces (Figure 4a). This seeding line can be changed instantly by choosing different isovalues a_0, b_0. Shape and location of the seeding line depend on $a_s(\mathbf{x})$ and $b_s(\mathbf{x})$ as well as the chosen isovalues. This gives a large amount of freedom for defining the seeding line and it can be placed anywhere in the domain.

A path surface describes the advection of the seeding line over time. We obtain it using the advected scalar fields $a(\mathbf{x}, t_s, \tau)$ and $b(\mathbf{x}, t_s, \tau)$. The intersection of the two isovolumes in 4D

$$[\, a(\mathbf{x}, t_s, \tau) = a_0 \;,\; b(\mathbf{x}, t_s, \tau) = b_0 \,] \tag{4}$$

is our implicit path surface. It is a 2D manifold in the 4D domain $D \times \Upsilon$. See Figure 5c.

This can also be seen less formally: the seeding line is the intersection of two isosurfaces of two 3D scalar fields $a_s(\mathbf{x})$ and $b_s(\mathbf{x})$ (Figure 4a). This is before any integration took place, i.e., $\tau_0 = 0$. Now we advect both scalar fields and arrive at τ_1. This advects the two isosurfaces and their intersection line as well (Figure 4b). Collecting the intersection lines over all τ_i gives us the path surface (Figure 4c).

The connection to time surfaces is now straightforward: an implicit path surface is obtained by repeatedly intersecting two implicit time surfaces over the course of their evolution. In fact, the two isovolumes in (4) represent two evolving time surfaces.

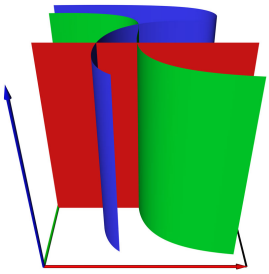
Again, we just defined an infinitely large family of path surfaces: changing the isovalues a_0, b_0 yields *instantly* a different path surface. See Figure 5d.

3.3. Implicit Stream Surfaces

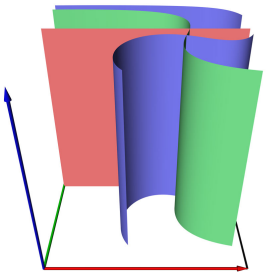
The setup from the last section can directly be applied to steady vector fields, where (4) describes an implicit *stream* surface. Our method is different than van Wijk's approach [vW93] and solves the three shortcomings discussed in Section 1:

- **Full domain coverage**
  Our seeding fields a_s and b_s are defined in the whole domain and not only at the boundary. Hence, we can create stream surfaces in all regions, even in the presence of closed orbits or vortex structures.
- **Multiple voxel intersections**
  Stream surfaces intersecting a voxel multiple times are accounted for by the additional τ-dimension.
- **Full control of the seeding line**
  The seeding line can be placed anywhere in the domain, not only at the boundary.

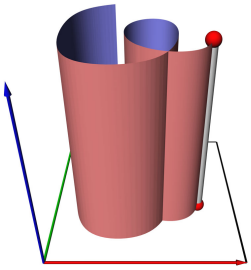
Figure 5e shows implicit stream surfaces obtained with our approach.



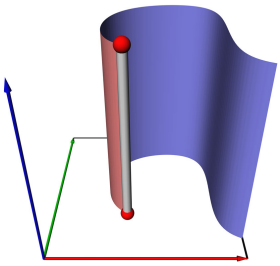
(a) Evolution of an implicit time surface from its seeding surface (red) over time (green, blue).



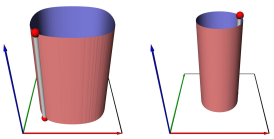
(b) A different time surface evolution is instantly obtained by selecting a different isovalue a_0.



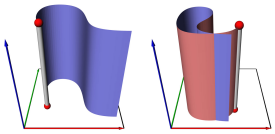
(c) Implicit path surface. Compare to Figure 2b. See also Figure 4.



(d) A different implicit path surface is instantly obtained by selecting different a_0, b_0.



(e) Implicit stream surfaces. Compare to Figure 2a (left).



(f) Implicit streak surfaces. Compare to Figure 2c.

Figure 5: *All four types of implicit integral surfaces in the example vector field. The time surfaces are given as temporal evolution of an isosurface through a 4D scalar field. Path, stream and streak surfaces are the intersections of two isovolumes in 4D. Seeding fields: $a_s(\mathbf{x}) = x$ and $b_s(\mathbf{x}) = y$.*

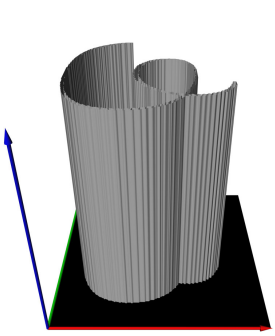
3.4. Implicit Streak Surfaces

A streak surface is the collection of all particles that went through a common seeding line at different times during their integration. More precisely, a streak surface is formed at a time t_e by time lines that have been seeded from the same spatial seeding line at different times.

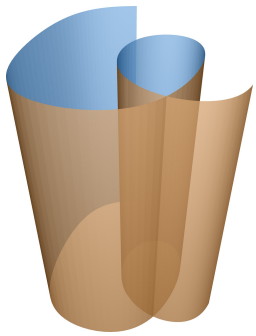
We describe the seeding line implicitly as we did it for path and stream surfaces: as an intersection of two isosurfaces of two seeding fields, see (3). The difference is in how we slice through the t, τ-dimensions of the advected scalar fields $a(\mathbf{x}, t, \tau)$ and $b(\mathbf{x}, t, \tau)$ from (1). In contrast to path surfaces, a streak surface gives the location of particles at a *constant* time step (Section 2.1). Denoting this time step as t_e, we are interested in all particles seeded at $t_s = t_e - \tau$ and integrated in the flow over the time interval τ. Hence, the implicit streak surface is given as the solution of

$$[\, a(\mathbf{x}, t_e - \tau, \tau) = a_0 \;,\; b(\mathbf{x}, t_e - \tau, \tau) = b_0 \,] \qquad (5)$$

Again, this is an intersection of two isovolumes in 4D, which gives a 2D manifold. As before, we just defined an infinitely large family of streak surfaces: changing the isovalues a_0, b_0 yields *instantly* a different streak surface. See Figure 5f.



(a) Projection into a 3D voxel grid.



(b) Raycasting in 4D.

Figure 6: *Visualization methods that exploit the implicit nature of the integral surfaces. Shown is a path surface from the example vector field – same data as in Figure 5c.*

4. Visualization Methods

Implicit time surfaces are the isosurfaces of the 3D scalar field (2). They can be visualized using any known volume visualization method such as direct volume rendering or Marching Cubes.

The other three implicit integral surfaces are the intersection of two isovolumes in 4D. We are not aware of any existing visualization method for this. In this section, we introduce three new visualization methods for this kind of data. The first two exploit the implicit nature of our integral surfaces: projection of the 4D field into a 3D volume (Section 4.1) and raycasting the 4D field directly (Section 4.2). The third visualization option allows to extract the surface as an explicit triangle mesh (Section 4.3).

For the sake of simplicity and to treat stream, path and streak surfaces alike, we will use t in the following to refer to the fourth dimension of the advected scalar fields. Hence, we want to visualize the implicit surface defined by

$$[\, a(\mathbf{x}, t) = a_0 \;,\; b(\mathbf{x}, t) = b_0 \,]. \qquad (6)$$

We assume the two advected scalar fields $a(\mathbf{x}, t)$ and $b(\mathbf{x}, t)$ to be piecewise quadro-linear fields sampled over an uniform 4D grid.

4.1. Projection into 3D

The probably simplest way to visualize the implicit surface defined by (6) is to project it into 3D on a per-voxel basis. To

do so, we initialize a 3D voxel grid with zeros. We call it *projected grid* and it has the same spatial dimensions as our 4D grid. Next, we visit each 4D voxel and check whether *both* isovolumes $a(\mathbf{x},t) = a_0$ and $b(\mathbf{x},t) = b_0$ run through this voxel. Checking for the existence of an isovolume through a 4D voxel is done by comparing the scalar values s_i at all 16 corners to the isovalue: there is no isovolume through this voxel, if all s_i are larger than the isovalue, or if all s_i are smaller than the isovalue. In all other cases, the isovolume runs through this voxel.

If both isovolumes run through a 4D voxel (i,j,k,m), we set the value "1" in the corresponding 3D voxel (i,j,k) of the projected grid. Obviously, this is only a rough approximation. The two isovolumes running through a 4D voxel may actually not intersect, i.e., our condition here is only necessary, but not sufficient. Furthermore, we loose subgrid accuracy due to the projection. However, the projection is very fast to compute since no interpolation takes place, and it is also very memory efficient since the projected grid just requires a bit field. Any volume visualization method can be used to render the projected grid. Figure 6a shows an example.

While this is not our visualization of choice, it does provide a quick preview. Furthermore, the projected grid serves as an acceleration data structure for the following two visualization methods.

4.2. Raycasting Implicit Surfaces in 4D

Consider a 4D unit cube $[0,1]^4$ with the two quadro-linearly interpolated scalar fields a,b. Let a ray

$$\mathbf{x}(s) = (1-s)\,\mathbf{x}_0 \;+\; s\,\mathbf{x}_1 \tag{7}$$

intersect the 3D spatial unit cube, where $\mathbf{x}_0$ and $\mathbf{x}_1$ are the entry and exit points, respectively. Then we search for our implicit surface (6) along the ray, i.e., we have to solve

$$[\, a(\mathbf{x}(s),t) = a_0 \;,\; b(\mathbf{x}(s),t) = b_0 \,] \tag{8}$$

for the two unknowns (s,t). Both equations in (8) are cubic in s, but only linear in t. Each of them can be solved for t, inserting this into the other one gives a sixth degree polynomial in s. The zeros of this polynomial are the intersections of the ray with our implicit surface. For each 3D voxel (i,j,k) intersected by the ray, we have to compute (8) for all corresponding 4D voxels $(i,j,k,m = 0 \ldots m_{max})$. For proper Phong lighting, we compute the normal of the implicit surface using the partial derivatives of the scalar fields a,b:

$$\mathbf{n}(\mathbf{x},t) = \begin{pmatrix} a_x b_t - a_t b_x \\ a_y b_t - a_t b_y \\ a_z b_t - a_t b_z \end{pmatrix}. \tag{9}$$

Implementation Details We send a ray from the camera through each pixel of the image plane into the 3D scene. If it hits the 3D bounding box of our data set, we traverse the ray through the volume, thereby visiting each 3D voxel intersected by the ray. This is done from front to back. Upon visiting a 3D voxel, we first look up its value in the projected grid (previous section). If the voxel is not set, then the implicit surface cannot go through it. This is a very fast test and it speeds up the raycasting a lot. If the voxel is set, however, we solve (8) for all corresponding 4D voxels through which the surface may run. In fact, we keep a list of such 4D voxels for each 3D voxel. It is obtained while computing the projected grid.

Solving (8) is straightforward with any standard polynomial root finder. We use the well-known Jenkins-Traub algorithm [Jen75] (the variation for real coefficients), which gave us stable results in all our examples and test cases.

Upon finding an intersection between ray and implicit surface, we compute the normal of the surface at this position using (9), which serves as input for the Phong lighting. Furthermore, this is also the moment in the algorithm, when any kind of color- or alpha-mapping should take place. For example, one could map the time-component as a color onto the surface. Finally, after traversing through the entire volume, we synthesize the final pixel color by alpha-blending all found intersections from back to front. Figure 6b shows a semi-transparent example with two-sided lighting.

4.3. Conversion to an Explicit Surface Representation

To convert our implicit surfaces to explicit triangle meshes, we follow the basic principles of the Marching Cubes algorithm [LC87]: it computes the intersection points of an implicit isosurface with the *edges* of a 3D voxel. The triangulation of these intersection points yields the explicit surface representation. Transferring this to 4D, we need to search for the intersection points of our implicit surface with the *boundary faces* of a 4D voxel.

We think of a 4D voxel as time-dependent 3D voxel and describe it as unit hypercube $[0,1]^4$ as shown in Figure 7. It has 24 boundary faces, each of them lies in either the xy-, xz-, yz-, xt-, yt-, or $zt-$plane, and the two remaining components are fixed to either 0 or 1. In Figure 7 for example, the yz-plane for $(x = 0, t = 0)$ is highlighted in green, and the xt-plane for $(y = 1, z = 1)$ is highlighted in red. Recall that Equation (6) describes the intersection of two time surfaces changing over time. This intersection gives a time-dependent 3D line structure. For a constant time, this line intersects a 3D voxel in two points and slices through it with increasing time. This evolution is illustrated in Figure 8.

The quadro-linear scalar fields $a(\mathbf{x},t)$ and $b(\mathbf{x},t)$ reduce to bi-linear fields when restricted to a boundary face of the 4D voxel, such that $a(\mathbf{x},t) = a_0$ and $b(\mathbf{x},t) = b_0$ describe two isolines. This simplifies (6) to a quadratic equation, which now describes the intersection of the two isolines on the boundary face. Solving this yields up to two isoline intersections per boundary face. We do this for all 24 boundary faces to find all points where our implicit surface intersects the 4D voxel.

Next, we search for a 3D triangulation of these points. We consider them as 3D time-dependent points on the faces of a

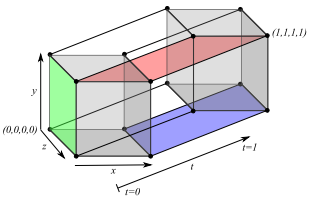


Figure 7: *4D hypercube with highlighted boundary faces.*

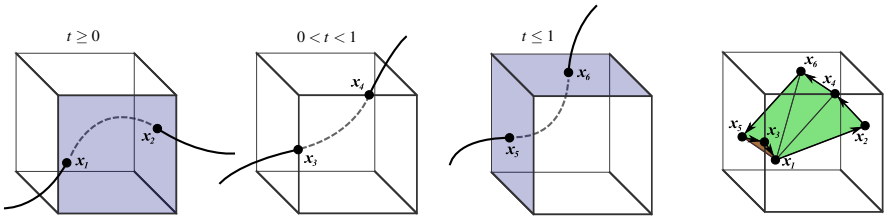
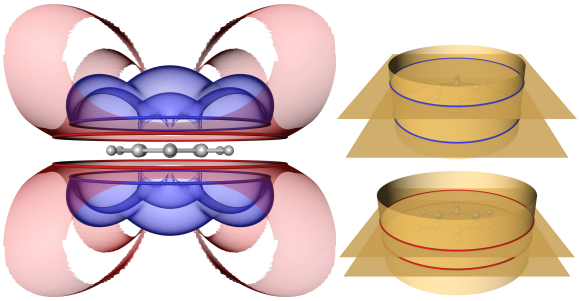


Figure 8: *Time-varying 3D line structure intersecting a 3D voxel over time, and corresponding triangulation.*



(a) Implicit stream surfaces. (b) Seeding fields.

Figure 10: *Implicit stream surfaces and their seeding fields in the steady Benzene data set.*

3D voxel. Note that all points found in the xt-, yt-, zt-planes live on the edges of the 3D voxel, while the ones found in the xy-, xz-, yz-planes live on its faces. Now we sort all the points in time and find the shortest circuit on the faces of the 3D voxel, as illustrated in Figure 8. Then we create triangle fans along this circuit to get the desired triangle mesh.

5. Applications

All our applications have been computed on a laptop with an Intel Xeon E31225 (3.1GHz) CPU and 8 GB RAM. This CPU has four cores and we used OpenMP to parallelize the computations.

Figure 9 shows the 3D unsteady flow around a confined square cylinder. This is a direct numerical Navier Stokes simulation by Camarri et al. [CSBI05]. The implicit path surface of this flow (Figure 9a) has been computed in forward and backward direction for $\tau = \pm 30$, starting from the depicted green seeding line at $t = 130$ using the seeding fields $a_s(\mathbf{x}) = x$ and $b_s(\mathbf{x}) = y$ with the isovalues $a_0 = 12$ and $b_0 = 1$. This is near the end of the flow simulation, where the flow is highly unsteady. Computing the flow map for this example took about 3 hours. Once this has been done, advecting the seeding fields is rather fast and took under a minute. In Figure 9a we rendered the path surface using an explicit triangle mesh. Converting the implicit representation took about 3 minutes.

Figure 9b shows implicit time surfaces of this flow. Rendering the time surfaces is very fast since it just involves computing isosurfaces of a 3D scalar field. We re-used one of the advected scalar fields from the path surface: this exemplifies that our approach is not limited to a certain type of integral surfaces. While it is expensive to compute the flow map, this is somewhat made up for by being able to switch between different integral surfaces at will.

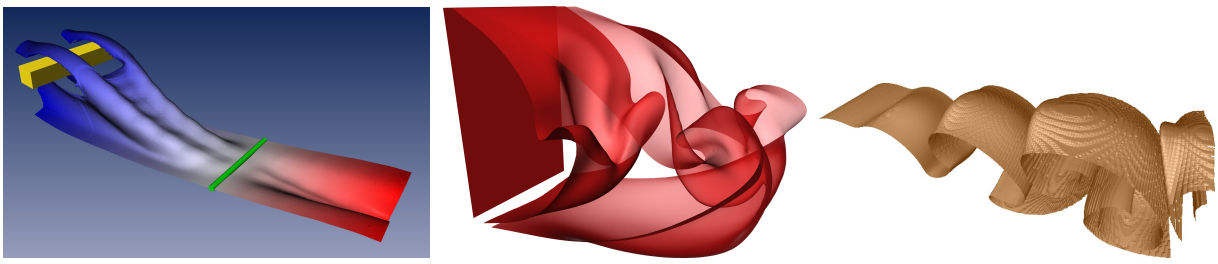
An example for our 4D raycaster is shown in Figure 9c, where a streak surface in the flow behind the square cylinder is shown. We needed to downsample the advected scalar fields drastically for this example, since our proof-of-concept raycasting implementation is not yet ready for prime time. In fact, it runs entirely on the CPU. We measured its performance for the shown streak surface and found that it traces 10k rays per second in a single thread, which makes about 40k rays per second with OpenMP support on our hardware.

Figure 10 visualizes the electrostatic field around a benzene molecule. It is a steady 3D vector field that exhibits 184 critical points. We chose this example to show that our method can deal with topologically complex 3D vector fields. Figure 10a shows four implicit stream surfaces. They have been computed using the same advected scalar fields, but with different isovalues a_0, b_0. Figure 10b shows the seeding fields $a_s(\mathbf{x}) = x^2 + y^2$ and $b_s(\mathbf{x}) = z^2$ as well as the intersections of their isosurfaces, which serve as seeding lines. Note that the two blue stream surfaces have been computed in one run: the chosen seeding field b_s allows to create two planes that intersect the cylindrical seeding field a_s. Similarly for the two red stream surfaces.

6. Conclusions and Future Work

We presented a novel method for generating and visualizing implicit integral surfaces in 3D time-dependent vector fields. Our implicit definition of integral surfaces provides the first single framework to create all four classic types of integral surfaces. Moreover, our approach offers a whole family of seeding structures and their corresponding integral surfaces at once. For visualizing our implicit integral surfaces, we described three approaches, two of which also exploit the implicit nature of our surfaces. We demonstrated our method for one steady and one unsteady example vector field, showing the four classic types of integral surfaces.

The method presented here requires a large amount of processing power due to the dense flow integration for pre-



(a) Implicit path surface in forward (red) and backward (blue) direction.

(b) Implicit time surfaces (forward) from $t = 130$ with $a_s(\mathbf{x}) = x$ and $a_0 = 5$.

(c) Implicit streak surface rendered using the 4D raycasting algorithm.

Figure 9: *Implicit integral surfaces in the square cylinder data set near the highly unsteady end of the flow simulation.*

computing the flow map. In terms of computational efficiency, our method cannot compete with explicit methods. However, the computation of the advected scalar fields yields much more information than a single integral surface. In fact, the advected scalar fields describe a family of surfaces from which the user can choose by changing the isovalues. For unsteady flows, this goes even one step further: two advected scalar fields suffice to describe path, streak, and time surfaces at the same time. This has been shown for the square cylinder data set in Figures 9c and 9.

Our method samples the domain to describe integral surfaces. Hence, sampling artifacts are unavoidable. Explicit methods do not suffer from that. At the same time, explicit methods require a careful implementation, since they need to take care of the underlying surface parametrization that is intrinsic to explicit methods. Our method, on the other hand, is straightforward to implement: it basically consists of particle integration, and root finding for the subsequent visualization. As such, it lends itself to massively parallel architectures. All operations can be carried out without any communication between parallel threads. For explicit methods, this is not necessarily the case: while some parts of these algorithms can be run in parallel, there is always a need for synchronization between threads; especially when it comes to taking care of the surface parametrization, e.g., when refining or coarsening the front line. Since hardware architectures of the future will rather be more parallel than they are already today, it seems reasonable to assume that massively parallel methods such as ours will benefit more from these developments.

References

[BFTW09] BÜRGER K., FERSTL F., THEISEL H., WESTERMANN R.: Interactive Streak Surface Visualization on the GPU. *IEEE TVCG (Proc. IEEE Visualization) 15*, 6 (2009), 1259–1266. 1

[BKP*10] BOTSCH M., KOBBELT L., PAULY M., ALLIEZ P., LEVY B.: *Polygon Mesh Processing*. AK Peters, 2010. 1

[CH97] CAI W., HENG P.-A.: Principal stream surfaces. In *Proc. IEEE Visualization* (1997), pp. 75–80. 2

[CKSW08] CUNTZ N., KOLB A., STRZODKA R., WEISKOPF D.: Particle level set advection for the interactive visualization of unsteady 3D flow. *Computer Graphics Forum (Eurovis) 27*, 3 (2008), 719–726. 2

[CSBI05] CAMARRI S., SALVETTI M.-V., BUFFONI M., IOLLO A.: Simulation of the three-dimensional flow around a square cylinder between parallel walls at moderate reynolds numbers. In *XVII Congresso di Meccanica Teorica ed Applicata* (2005). 7

[GKT*08] GARTH C., KRISHNAN H., TRICOCHE X., TRICOCHE T., JOY K. I.: Generation of accurate integral surfaces in time-dependent vector fields. *IEEE TVCG (Proc. IEEE Visualization) 14*, 6 (2008), 1404–1411. 1

[Hal01] HALLER G.: Distinguished material surfaces and coherent structures in three-dimensional fluid flows. *Physica D 149*, 4 (2001), 248–277. 3

[Hul92] HULTQUIST J.: Constructing stream surfaces in steady 3D vector fields. In *Proc. IEEE Visualization* (1992), pp. 171–177. 1

[Jen75] JENKINS M. A.: Algorithm 493: Zeros of a real polynomial [c2]. *ACM Trans. Math. Softw. 1*, 2 (June 1975), 178–189. 6

[KGJ09] KRISHNAN H., GARTH C., JOY K.: Time and streak surfaces for flow visualization in large time-varying data sets. *IEEE TVCG (Proc. IEEE Visualization) 15*, 6 (2009), 1267–1274. 1

[KM92] KENWRIGHT D. N., MALLINSON G. D.: A 3-d streamline tracking algorithm using dual stream functions. In *Proc. IEEE Visualization* (1992), pp. 62–68. 2

[LC87] LORENSEN W., CLINE H. E.: Marching cubes: a high resolution 3D surface reconstruction algorithm. *Computer Graphics 21* (1987), 163–169. 6

[PS09] PEIKERT R., SADLO F.: Topologically relevant stream surfaces for flow visualization. In *Proc. SCCG* (2009), pp. 43–50. 1

[SBM*01] SCHEUERMANN G., BOBACH T., MAHROUS H. H. K., HAMANN B., JOY K., KOLLMANN W.: A tetrahedra-based stream surface algorithm. In *Proc. IEEE Visualization* (2001), pp. 151–158. 1

[Sha05] SHADDEN S. C.: Lagrangian coherent structures. http://mmae.iit.edu/shadden/LCS-tutorial/, 2005. 2

[Sta98] STALLING D.: *Fast Texture-based Algorithms for Vector Field Visualization*. PhD thesis, FU Berlin, Department of Mathematics and Computer Science, 1998. 1

[vFWTS08] VON FUNCK W., WEINKAUF T., THEISEL H., SEIDEL H.-P.: Smoke surfaces: An interactive flow visualization technique inspired by real-world flow experiments. *IEEE TVCG (Proc. IEEE Visualization) 14*, 6 (2008), 1396–1403. 1

[vW93] VAN WIJK J.: Implicit stream surfaces. In *Proc. IEEE Visualization* (1993), pp. 245–252. 1, 2, 4

[Wei04] WEISKOPF D.: Dye Advection Without the Blur: A Level-Set Approach for Texture-Based Visualization of Unsteady Flow. *Computer Graphics Forum (Proc. Eurographics) 23*, 3 (2004), 479–488. 2

[WHT12] WEINKAUF T., HEGE H.-C., THEISEL H.: Advected tangent curves: A general scheme for characteristic curves of flow fields. *Computer Graphics Forum (Proc. Eurographics) 31*, 2 (2012), 825–834. 3

[WJE00] WESTERMANN R., JOHNSON C., ERTL T.: A level-set method for flow visualization. In *Proc. IEEE Visualization* (2000), pp. 147–154. 2

[WT10] WEINKAUF T., THEISEL H.: Streak lines as tangent curves of a derived vector field. *IEEE TVCG (Proc. IEEE Visualization) 16*, 6 (2010), 1225–1234. 3