UC Davis
IDAV Publications

Title
A Tetrahedra-Based Stream Surface Algorithm

Permalink
https://escholarship.org/uc/item/86z9x1cc

Authors
Scheuermann, Gerik
Bobach, T.
Hagen, Hans
et al.

Publication Date
2001

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

A tetrahedra-based stream surface algorithm

Gerik Scheuermann[1,2] Tom Bobach[1] Hans Hagen[1] Karim Mahrous[2] Bernd Hamann[2]
Kenneth I. Joy[2] Wolfgang Kollmann[3]

Abstract

This paper presents a new algorithm for the calculation of stream surfaces for tetrahedral grids. It propagates the surface through the tetrahedra, one at a time, calculating the intersections with the tetrahedral faces. The method allows us to incorporate topological information from the cells, e.g., critical points. The calculations are based on barycentric coordinates, since this simplifies theory and algorithm. The stream surfaces are ruled surfaces inside each cell, and their construction is started with line segments on the faces. Our method supports the analysis of velocity fields resulting from computational fluid dynamics (CFD) simulations.

Keywords: vector field visualization, flow visualization, tetrahedral grid, unstructured grid, flow surface

1 Introduction

We are interested in the structure of 3D vector fields, especially the velocity fields of fluid flows. Most visualizations use streamlines to look for flow field structures. More sophisticated methods analyze the field by searching for line-type features [7, 10]. One can obtain substantial information from these visualizations, but many unsolved questions remain. We strongly believe that these problems stem partly from the concentration on streamlines. Stream surfaces provide much more information if one uses a precise algorithm for their calculation. A former algorithm from Hultquist [6] yields good results if the stream surface does not enter regions with strong twist or related problems that result in folding of the surface. Unfortunately, these complicated regions are typically of major interest in fluid dynamics. This paper presents a new algorithm for stream surfaces in tetrahedral grids. Our algorithm assumes piecewise linear interpolation over the tetrahedral grid. This assumption allows us to precisely describe the surface in each cell, since there is a cellwise analytic description for the whole stream surface. Therefore, we can also control the local error of stream surfaces and incorporate information about the flow structure in the cells. This yields more precise calculations than previous approaches, which is especially important for the calculation of stream surfaces approaching vortices. Figure 1 illustrates the basic idea for the stream surface construction inside a single tetrahedron.

[1]Computer Science Department, University of Kaiserlautern, PO Box 3049, 67653 Kaiserslautern, Germany, {scheuer,tbobach,hagen}@informatik.uni-kl.de

[2]Center for Image Processing and Integrated Computing (CIPIC), Department of Computer Science, University of California, Davis, CA 95616-8562, {mahrous,hamann,kijoy}@cs.ucdavis.edu

[3]Department of Mechanical and Aeronautical Engineering, University of California, Davis, CA 95616, {wkollmann}@ucdavis.edu

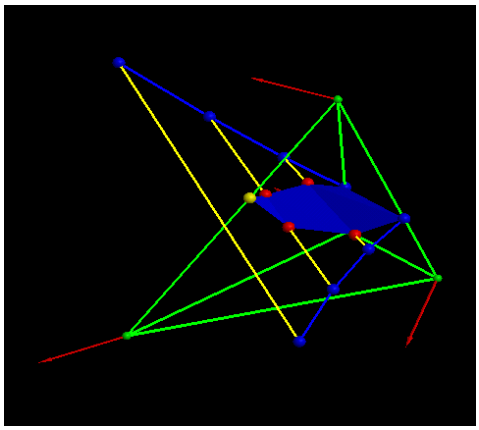

Figure 1: Stream surface construction inside tetrahedron. The endpoints are traced as streamlines. Each intermediate point is connected to its counterpart by a line on the surface. Clipping against the tetrahedron's faces yields the surface.

2 Stream Surfaces of Piecewise Linear Vector Fields in 3D Space

Not many papers exist concerning the construction of stream surfaces. At least this is true for the visualization research community. Hultquist [6] traces multiple streamlines and connects calculated points on the lines by triangles. Since converging and diverging flow presents a major difficulty for this procedure, he introduces additional points in case of diverging neighboring streamlines and removes traces in converging flow. Van Wijk [13] creates implicit stream surfaces by calculating streamlines starting at all grid points. He assigns values to the stream lines in a region of interest defining a scalar function that is constant on the stream lines so that one can obtain stream surfaces as isosurfaces of this function. He presents a second construction in the same paper based on the convection equation. This requires a large number of calculations and setting the right values in a region of interest. Our approach is more related to Hultquist's algorithm: We are also using a parametric approach to define the stream surface.

In the following, we shall denote vector fields, vectors, and points using 3D Cartesian coordinate notation. For example, $\bar{v} \in I\!R^3$ denotes a vector in 3D space. Thus, we can distinguish these quantities from their corresponding barycentric coordinate tupels that we view as elements in 4D space. For example, $v \in I\!R^4$ denotes the 4D barycentric parameter tuple associated with the vector $\bar{v} \in I\!R^3$. Scalar parameters are written as greek symbols, e.g., τ denotes time, σ denotes the 1D barycentric parameter along a line segment, and λ_1, λ_2 and λ_3 represent the eigenvalues of the Jacobian of a vector field. We review some necessary definitions and theorems. Let $D \subset I\!R^3$ be the domain. A vector field on D is a

map

$$\begin{aligned} \bar{v} : D &\rightarrow \mathbb{R}^3, \\ \bar{x} &\mapsto \bar{v}(\bar{x}). \end{aligned} \tag{1}$$

Physicists and engineers are mainly interested in the stream lines and surfaces of a vector field. A stream line passing through a point $\bar{x} \in D$ of $\bar{v}$ is a continuous map

$$\bar{s}_{\bar{x}} : J \rightarrow \mathbb{R}^3 , \tag{2}$$

where $0 \in J \subset \mathbb{R}$ is an interval and

$$\bar{s}_{\bar{x}}(0) = \bar{x} \quad \text{and} \quad \dot{\bar{s}}_{\bar{x}}(\tau) = \bar{v}\big(\bar{s}_{\bar{x}}(\tau)\big) \quad \forall\, \tau \in J. \tag{3}$$

A stream surface is defined as the union of all streamlines through the points of a generating curve $\bar{c} : I \rightarrow \mathbb{R}^3$, $\sigma \mapsto \bar{c}(\sigma)$ where $I \subset \mathbb{R}$ is an interval. In parametric form we get

$$\begin{aligned} \bar{f}_{\bar{c}} : I \times J &\rightarrow \mathbb{R}^3, \\ (\sigma, \tau) \mapsto \bar{f}_{\bar{c}}(\sigma, \tau), & \end{aligned} \tag{4}$$

with the conditions

$$\bar{f}_{\bar{c}}(\sigma, 0) = \bar{c}(\sigma) \tag{5}$$

$$\frac{\partial}{\partial \tau} \bar{f}_{\bar{c}}(\sigma, \tau) = \bar{v}(\bar{f}_{\bar{c}}(\sigma, \tau)) \tag{6}$$

Existence and uniqueness can be proved provided that $\bar{v}$ is Lipschitz-continuous. We consider only piecewise linear vector fields. One reason for this restriction is the existence of exact analytic solutions for the stream lines for linear elements. This fact has been used before [8, 9, 11]. Since each linear tetrahedral element of the field is defined by four values $\bar{v}_0, \ldots, \bar{v}_3 \in \mathbb{R}^3$ at the four vertices $\bar{p}_0, \ldots, \bar{p}_3 \in \mathbb{R}^3$, we can use barycentric coordinates as was done by Nielson et al. [9]. A linear vector field is defined on the hyperplane

$$\mathbb{B}^3 = \left\{ x \in \mathbb{R}^4 \,\middle|\, \sum_i x_i = 1 \right\} \subset \mathbb{R}^4 \tag{7}$$

by the four values

$$v(1,0,0,0) = v_0 = (\nu_0^0, \ldots, \nu_0^3) , \tag{8}$$

$$v(0,1,0,0) = v_1 = (\nu_1^0, \ldots, \nu_1^3) , \tag{9}$$

$$v(0,0,1,0) = v_2 = (\nu_2^0, \ldots, \nu_2^3) \text{, and} \tag{10}$$

$$v(0,0,0,1) = v_3 = (\nu_3^0, \ldots, \nu_3^3) . \tag{11}$$

These values are computed by adding each velocity vector to its corresponding vertex, computing the barycentric coordinates with respect to all four vertices $\bar{p}_0, \bar{p}_1, \bar{p}_2, \bar{p}_3 \in \mathbb{R}^3$, and finally substracting the barycentric coordinates of the corresponding vertex, i.e.,

$$\bar{v}_i + \bar{p}_i = \sum_{j=0}^{3} \left(\nu_i^j + \delta_i^j\right) \bar{p}_j , \tag{12}$$

where δ_i^j is the Kronecker delta and $\sum_{j=0}^{3} \left(\nu_i^j + \delta_i^j\right) = 1$. Since the vector field is parallel to $\mathbb{B}^3$ in $\mathbb{R}^4$, we set

$$v(0,0,0,0) = \mathbf{0} \tag{13}$$

and define the linear vector field as

$$\begin{aligned} v : \mathbb{R}^4 &\rightarrow \mathbb{R}^4 \\ x &\mapsto \begin{pmatrix} v_0 & v_1 & v_2 & v_3 \end{pmatrix} x = V x. \end{aligned} \tag{14}$$

This corresponds to the affine linear vector field

$$\begin{aligned} \bar{v} : \mathbb{R}^3 &\rightarrow \mathbb{R}^3 \\ \bar{x} &\mapsto A\bar{x} + \bar{b} , \end{aligned} \tag{15}$$

where $\bar{v}(\bar{p}_0) = \bar{v}_0, \ldots,$ and $\bar{v}(\bar{p}_3) = \bar{v}_3$. We are concerned with the stream lines

$$\begin{aligned} s_a : J &\rightarrow \mathbb{B}^3 , \\ \tau &\mapsto s_a(\sigma, \tau) , \end{aligned} \tag{16}$$

where

$$s_a(0) = a \quad \text{and} \tag{17}$$

$$\dot{s}_a(\tau) = v\big(s_a(\tau)\big) \quad . \tag{18}$$

The solution to this initial value problem is given by

$$s_a(\tau) = e^{V\tau} a, \tag{19}$$

see [2, 5]. For a stream surface

$$\begin{aligned} f_c : J \times I &\rightarrow \mathbb{B}^3, \\ (\sigma, \tau) &\mapsto f_c(\sigma, \tau), \end{aligned} \tag{20}$$

we obtain

$$f_c(\sigma, \tau) = e^{V\tau} c(\sigma) . \tag{21}$$

One can prove that V has the same eigenvalues as A, and an additional eigenvalue that is zero. When a vector field has a critical point in 3D space, the position vector of this critical point in $\mathbb{B}^3$ is an eigenvector associated with the additional zero eigenvalue. The normal form of V allows a direct computation of $e^{V\tau}$, and 21 cases are possible. The first two cases are common. All other cases are exceptional cases characterized by A having a zero eigenvalue or two equal eigenvalues. We give the formulae for the two common cases. The remaining formulae can be easily derived from the formulae below and Nielson's paper [9]. The first case C1 is characterized by three different eigenvalues λ_1, λ_2, and λ_3 and can be solved by the following formulae:

$$(C1) \quad V = S \begin{pmatrix} 0 & & & \\ & \lambda_1 & & \\ & & \lambda_2 & \\ & & & \lambda_3 \end{pmatrix} S^{-1} ,$$

$$e^{V\tau} = S \begin{pmatrix} 1 & & & \\ & e^{\lambda_1 \tau} & & \\ & & e^{\lambda_2 \tau} & \\ & & & e^{\lambda_3 \tau} \end{pmatrix} S^{-1} ,$$

$$f_c(\sigma, \tau) = \left(e^{\lambda_1 \tau} - 1\right) e_1 + (e^{\lambda_2 \tau} - 1) e_2 + \left(e^{\lambda_3 \tau} - 1\right) e_3 + c(\sigma) ,$$

$$e_1 = \frac{V}{\lambda_1} \left(\frac{V - \lambda_2 I}{\lambda_1 - \lambda_2}\right) \left(\frac{\lambda_3 I - V}{\lambda_3 - \lambda_1}\right) c(\sigma) ,$$

$$e_2 = \frac{V}{\lambda_2} \left(\frac{V - \lambda_3 I}{\lambda_2 - \lambda_3}\right) \left(\frac{\lambda_1 I - V}{\lambda_1 - \lambda_2}\right) c(\sigma) , \text{ and}$$

$$e_3 = \frac{V}{\lambda_3} \left(\frac{V - \lambda_1 I}{\lambda_3 - \lambda_1}\right) \left(\frac{\lambda_2 I - V}{\lambda_2 - \lambda_3}\right) c(\sigma) .$$

$$S = \begin{pmatrix} z & e_1 & e_2 & e_3 \end{pmatrix}$$

z denotes the critical point. The dependence of the vectors $e_1, e_2,$ and e_3 on $c(\sigma)$ is limited to the magnitude, since the vectors e_i are eigenvectors of the corresponding eigenvalues. Figure 2 illustrates

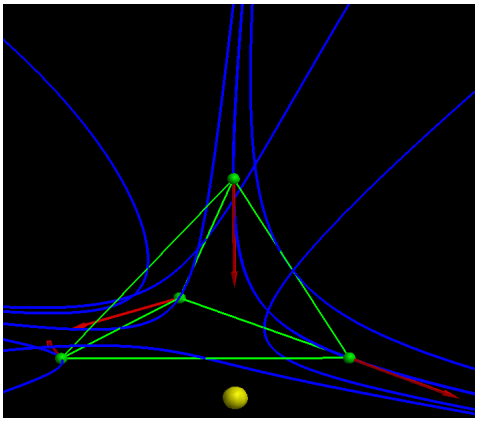

Figure 2: Vector field for case C1 with different signs of eigenvalues; the critical point outside the tetrahedron is shown to simplify interpretation.

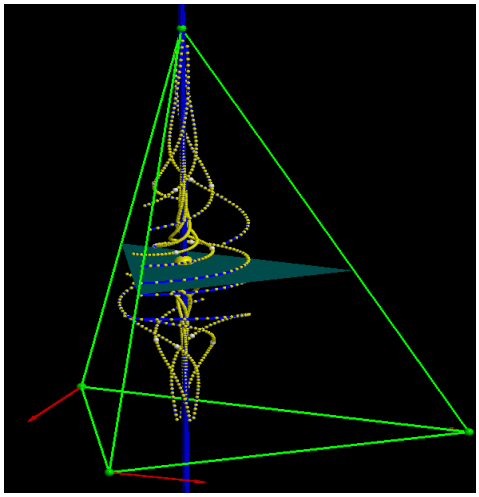

Figure 3: Vector field for case C2 exhibiting spiraling behavior around axis given by eigenspace of real eigenvalue. Critical point and rotation axis are stream lines themselves.

the vector field in this case by drawing streamlines. If all eigenvalues have the same sign, all stream lines start or end at the critical point.

The second case is characterized by a complex-conjugate pair of eigenvalues and one real eigenvalue. Complex eigenvalues indicate rotations, as shown in Figure 3. The corresponding formulae for case C2 are:

$$
\begin{aligned}
(C2)\ V &= S \begin{pmatrix} 0 & & \\ & \lambda_1 & \\ & & \begin{matrix} \lambda & -\mu \\ \mu & \lambda \end{matrix} \end{pmatrix} S^{-1}\,, \\
e^{V\tau} &= S \begin{pmatrix} 1 & & \\ & e^{\lambda_1\tau} & \\ & & \begin{matrix} e^{\lambda\tau}\cos(\mu\tau) & -e^{\lambda\tau}\sin(\mu\tau) \\ e^{\lambda\tau}\sin(\mu\tau) & e^{\lambda\tau}\cos(\mu\tau) \end{matrix} \end{pmatrix} S^{-1} \\
f_c(\sigma,\tau) &= \left(e^{\lambda_1\tau}-1\right) e_1 + \left(e^{\lambda\tau}\cos(\mu\tau)-1\right) e_2 + \\
&\quad e^{\lambda\tau}\sin(\mu\tau) e_3 + c(\sigma)\,, \\
e_1 &= \frac{V}{\lambda_1}\left(\frac{V^2 - 2\lambda V + (\lambda^2+\mu^2)I}{\lambda_1^2 - 2\lambda\lambda_1 + (\lambda^2+\mu^2)}\right) c(\sigma)\,, \\
e_2 &= V\left(\frac{(2\lambda-\lambda_1)V^2 + (\lambda_1^2 - 3\lambda^2 + \mu^2)V}{(\lambda^2+\mu^2)\left(2\lambda\lambda_1 - \lambda_1^2 - (\lambda^2+\mu^2)\right)} + \right. \\
&\quad \left. \frac{\lambda_1(3\lambda^2 - \mu^2 - 2\lambda\lambda_1)I}{(\lambda^2+\mu^2)\left(2\lambda\lambda_1 - \lambda_1^2 - (\lambda^2+\mu^2)\right)}\right) c(\sigma)\,, \\
e_3 &= V\left(\frac{(\lambda\lambda_1 - \lambda^2 + \mu^2)V^2 + (\lambda^3 - 3\lambda\mu^2 - \lambda\lambda_1^2)}{\mu(\lambda^2+\mu^2)\left(2\lambda\lambda_1 - \lambda_1^2 - (\lambda^2+\mu^2)\right)} + \right. \\
&\quad \left. \frac{(\lambda_1^2(\lambda^2-\mu^2) - \lambda_1(\lambda^3 - 3\lambda\mu^2))I}{\mu(\lambda^2+\mu^2)\left(2\lambda\lambda_1 - \lambda_1^2 - (\lambda^2+\mu^2)\right)}\right) c(\sigma)\,. \\
S &= \begin{pmatrix} z & e_1 & e_2 & e_3 \end{pmatrix}
\end{aligned}
$$

Similarly to case C1, e_1 is an eigenvector associated with the real eigenvalue λ_1. The vectors e_2 and e_3 lie in the eigenplane of the two complex eigenvalues $\lambda \pm i\mu$. Figure 3 illustrates the spiraling motion around the e_1-axis by drawing streamlines.

We are concerned with stream surfaces in a tetrahedral grid. Such a stream surface is defined by a curve

$$
\begin{aligned}
\bar{c} : I\!R \supset I &\rightarrow I\!R^3, \\
\sigma &\mapsto \bar{c}(\sigma).
\end{aligned} \tag{22}
$$

Since we have an analytic formula for a stream surface in each single tetrahedron $\bar{T}$, we intersect the curve $\bar{c}$ with the tetrahedron:

$$
\begin{aligned}
\bar{c}_{\bar{T}} : I \supset I_T &\rightarrow \bar{T}, \\
\sigma &\mapsto \bar{c}(\sigma).
\end{aligned} \tag{23}
$$

Then, we transform $\bar{c}_{\bar{T}}$ in barycentric coordinates with respect to $\bar{T}$ and call it c_T.

$$
\begin{aligned}
c_T : I_T &\rightarrow I\!B^3, \\
\sigma &\mapsto c(\sigma).
\end{aligned} \tag{24}
$$

As stream surface, we find

$$
\begin{aligned}
f_T : I_T \times J \supset D_T &\rightarrow I\!B^3, \\
(\sigma,\tau) &\mapsto e^{V\tau} c(\sigma).
\end{aligned} \tag{25}
$$

(We have to limit $I_T \times J$ to D_T to keep f_T inside the tetrahedron.) In our barycentric coordinates, we have

$$
T = \{x \in I\!B^3 | x_j \geq 0, j = 0,1,2,3\} \tag{26}
$$

and for the faces F_i, $i = 0,1,2,3$,

$$
F_i = \{x \in I\!B^3 | x_i = 0, x_j \geq 0, j \neq i\}\,. \tag{27}
$$

Therefore, we have to choose D_i such that $f_T(D_T) \subset \{x \in I\!B^3 | x_j \geq 0, j = 0,1,2,3\}$.

Since we are interested in the whole stream surface in the grid, we have to calculate the **exit curves** where the stream surface leaves the tetrahedron. These curves are the starting curves for the surface in the neighboring tetrahedra. For the exit curves, we have

$$
\begin{aligned}
d_{T,i} : I_T \supset I_{T,i} &\rightarrow F_i \subset I\!B^3, \\
\sigma &\mapsto f_T(\sigma, \tau_\sigma).
\end{aligned} \tag{28}
$$

τ_σ is the smallest value such that the trace for σ leaves the tetrahedron, i.e. $\tau_\sigma := \min\{\tau | f_T(\sigma,\tau) \in T, \exists \epsilon > 0 \forall 0 < \delta < \epsilon : f_T(\sigma,\tau+\delta) \notin T\}$. I_{T_i} is the set of subintervals of I where the trace leaves the tetrahedron through face F_i, so $I_{T_i} := \{\sigma \in I_T | f_T(\sigma,\tau_\sigma) \in F_i\}$. From the formula for the stream surface above we see that we have only an implicit definition of τ_σ which we will solve only approximately in the algorithm.

If T_i is the neighboring tetrahedron along face F_i of T, $d_{T,i}$ describes the starting curve for the continuation of the surface through T_i. In this way, we can propagate the surface through the whole mesh. The propagation ends if the surface reaches the boundary of the mesh or a critical point. If the critical point is a sink, at least part of the stream surface will end there. It is possible that the surface is split here in two parts that are propagated further and a third part in the middle that ends at the sink. The next section describes the necessary approximation of all curves by polylines and resulting simplifications in the algorithm.

3 Linear Stream Surface Calculation

We assume that the generating curve $\bar{c}$ consists of connected line segments which is quite typical for a visualization application. Since the calculation treats each tetrahedral cell separately, we have to intersect the generating line segments with the tetrahedra. For the calculation, we focus our description now on a single line segment in a single tetrahedron. Let

$$\begin{aligned} c : [0,1] &\rightarrow \mathbb{B}^3, \\ \sigma &\mapsto (1-\sigma)a + \sigma b \end{aligned} \tag{29}$$

be the generating line segment. A key idea of our algorithm is that the stream surface defined by the line segment is

$$\begin{aligned} F : [0,1] \times \mathbb{R}^3 \supset D_T \rightarrow \mathbb{B}^3, \\ (\sigma,\tau) \mapsto (1-\sigma)e^{V\tau}a + \sigma e^{V\tau}b. \end{aligned} \tag{30}$$

This is valid because for stream lines in linear vector fields holds

$$s_{(1-\sigma)a+\sigma b}(\tau) = (1-\sigma)s_a(\tau) + \sigma s_b(\tau) \tag{31}$$

which can be checked by directly using equations (3). Formula (30) describes a ruled surface, i.e. a surface that is generated by two parametric curves blended by line segments. This can be seen easily, since for fixed τ, we get a line segment with parameter σ. We compute the stream lines (traces) through points a and b by stepping forward with a fixed stepsize $\delta\tau$. We set $\tau_i = i\Delta\tau$. By connecting $s_a(\tau_i)$ and $s_b(\tau_i)$, we obtain line segments lying on the actual stream surface. If the trace through a or b leaves the tetrahedron, we call this an **exit point**, see Figure 4. An exit point marks the begin of an exit curve. If the trace through a or b is outside the tetrahedron, we clip the line segment against the faces yielding line segments on the stream surface inside the tetrahedron. Since the faces F_i of the tetrahedron are defined by a zero in the i-th coordinate (see previous section), we only need to check for zero crossings to obtain the line segment inside a tetrahedron. Since we need the intersections of the stream surface with the edges as end points of the exit curves, we have to find these **split points**, see Figure 5. The whole process is illustrated in Figure 6.

But there is an exception to this rule: We are not allowed to define points on an inflow part of a face. Obviously, no part of the stream surface can leave the tetrahedron here. The problem arises if on a boundary face, a line (**switch line**) with flow vectors tangent to the face can be found, see Figure 7. Therefore, we define points on the outflow part as described above, but once we cross the switch line, we compute the intersection of the stream surface with this line and call the intersection **switch point**, see Figure 8. For τ larger than the parameter for the switch point, we have to replace a resp. b with the switch point. (In the unlikely case of parts of the switch line being part of the stream surface, we take the point where we leave the switch line as switch point.) The exit curves consist of all the segments generated on the faces of the tetrahedra. We will see in the next section that one has to remove too small segments to avoid a fast growing number of segments.

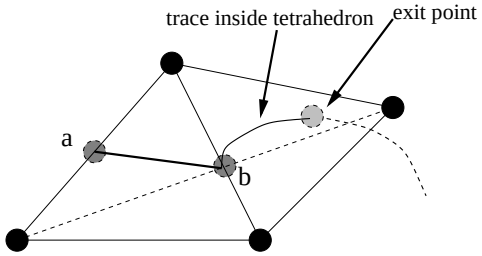

Figure 4: An exit point is a point where a trace leaves a tetrahedron.

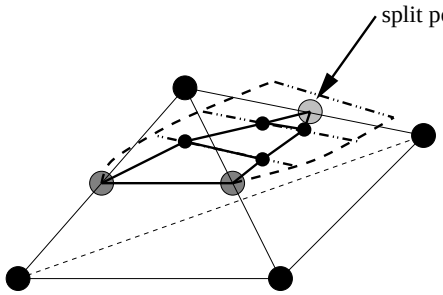

Figure 5: A split point is a point where a stream surface intersects an edge of a tetrahedron.

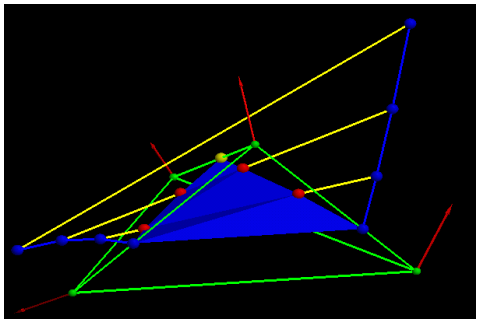

Figure 6: Stream surface defined by a line segment (ruled surface) inside a tetrahedron. We advect end points of the segment through the flow and connect them by lines. After clipping the lines against the faces of the tetrahedron, we triangulate the stream surface.

4 Algorithm

Our algorithm requires the propagation of stream surfaces through a tetrahedral grid. In contrast to very general methods like the method from Stämpfle [12], it is advantageous to use the grid structure and its influence on the vector field directly to control error.

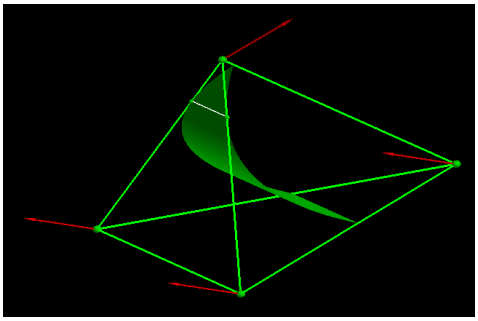

Figure 7: A switch line is defined as border between inflow and outflow on a tetrahedron face. As a result of linear interpolation, there can be only one switch line per face.

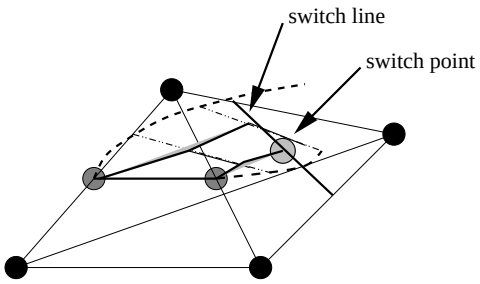

Figure 8: At a switch point a surface intersects a switch line. The trace must be continued from the switch point.

Previous visualization methods, see for example [4], [6], demonstrate the problems arising from "folding", "turning", "widening", and "stretching" of stream surfaces. Our method overcomes these difficulties naturally by using the understanding of the affine linear flow in our tetrahedral cells. We now describe our data structure and algorithm.

4.1 Representation of the Surface

Our representation of the surface is based on a two-step construction. First, we keep an advancing front of line **segments** and parts of stream lines, called **traces**. This approach models the boundary of the surface to be constructed at every moment and is used to extend the surface to its final form. Second, we define a triangulation of the surface for the OpenGL rendering pipeline. This triangulation is the approximation of the constructed part of the surface. Similar ideas have been introduced by Hultquist [6]. The basic difference is that our algorithm uses the grid and the topological information about the structure of linear vector fields to solve the problems of folding, converging, diverging, and winding in an accurate manner.

Triangles are generated and passed to the OpenGL rendering pipeline. We focus our description on the advancing front. The front is implemented as a double-linked list of points, where the connections are marked as "line segment," "forward trace," or "backward trace" with respect to the point that carries the label. We store, for each connection, the tetrahedron the front is about to penetrate (either in flow direction or reverse flow direction, depending on the direction of surface integration). We describe in the next sub-section how the links provide all the information necessary for surface propagation and updating the front. For practical purposes, we store only the segments that will be advanced through the field and the traces between these segments. Thus, when a segment approaches the boundary, we remove the corresponding connection from the list. At first glance, it might appear strange that we allow the traces to be part of the front, since the flow will usually be transversal to the generating polyline c, and we will not have any traces. We discuss in the following sub-section that traces are created naturally by our algorithm, and we need to keep them to determine when one can remove a point from the front. We attempt to minimize the number of segments, considering the fact that we might be dealing with massive data sets.

4.2 Segment Propagation

The basic step of our algorithm advances a line segment

$$|\overline{ab}| = \{(1-\sigma)a + \sigma b | \sigma \in [0,1]\} \tag{32}$$

through a tetrahedron T. Three tasks have to be accomplished:

(i) Calculate the stream surface as described in Section 3.

(ii) Create the triangles for the OpenGL rendering pipeline.

(iii) Update the advancing front.

Before we start, we calculate the switch lines on the faces (where the flow is tangential to the face). For face F_i, the intersection x of the split line with edge $k \neq i$ is defined by

$$v(x)_i = (Vx)_i = \sum_{j=0}^{3} v_{ij} x_j \quad \text{with } x_k = x_i = 0\,. \tag{33}$$

This leads to 12 linear equations for all edges of all faces and 24 comparisons with zero to determine whether or not the point is on the actual edge or only on the line through that edge.

After this, we start the integration at the points $a^0 := a$ and $b^0 := b$ using uniform time steps $\Delta\tau$. With $M := e^{V\Delta\tau}$, we obtain series of points

$$a^n = M^n a^0 \qquad b^n = M^n b^0 \tag{34}$$

defining line segments on the exact stream surface. For the trace a and b, each point has to be clipped against the terahedron. Besides, we have to find exit points, split points, switch points and figure out that the whole segment has left the tetrahedron. The following pseudo-code describes this process in detail.

```
// The following variables need valid values
bool trace_a_was_outside_tet;
bool trace_b_was_outside_tet;
int a_was_on_face;
int b_was_on_face;
// Calculate the next segment
a = M * a;
b = M * b;
// Clip against the tetrahedron
k = 0;
l = 1;
FOR (i=0; i < 4; i++)
  IF (a[i] < 0)
    trace_a_outside_tet = TRUE;
    IF (b[i] < 0)
      line_outside_tet = TRUE;
    ELSE
      temp = a[i] / (a[i] - b[i]);
      IF (temp > k)
        k = temp;
        a_on_face = i;
```

```
      END IF
    END IF
  ELSE IF (b[i] < 0)
    trace_b_outside_tet = TRUE;
    temp = a[i] / (a[i] - b[i]);
    IF (temp < l)
      l = temp;
      b_on_face = i;
    END IF
  END IF
END FOR
aa = (1 - k) a + k b;
bb = (1 - l) a + l b;
// Check for line outside tet
IF (k > l) line_outside_tet = TRUE;
// Check for exit points and split points
IF trace_a_outside_tet
  IF NOT trace_a_was_outside_tet
    calulate exit point;
  ELSE IF (a_was_on_face != a_on_face)
    calculate split point;
  END IF
END IF
IF trace_b_outside_tet
  IF NOT trace_b_was_outside_tet
    calulate exit point;
  ELSE IF (b_was_on_face != b_on_face)
    calculate split point;
  END IF
END IF
// Check for switch points
IF trace_a_outside_tet
  v = V aa;
  IF (v[a_on_face] > 0)
    calculate switch point;
END IF
IF trace_b_outside_tet
  v = V bb;
  IF (v[b_on_face] > 0)
    calculate switch point;
END IF
```

The split points define new segments in the front since they define the line segment that has to be propagated through a neighboring tetrahedron. The switch points are also defining new segments. We have to follow the two traces until the connecting line segment between two points of the same time is completely outside the tetrahedron. Then we create the triangles for the OpenGL rendering pipeline. We start with a^0 and b^0, add $a^1, \ldots$, $b^1, \ldots$, exit points, split points, and switch points from both traces in the order of their creation into a triangle strip (for reduction of points, see sub-section 4.5).

Finally, we have to update the surface front representation. We have to add all the exit points, split points, and switch points together with the connecting traces and new segments. Traces are created by integrating inside a tetrahedron (either at the beginning or after a switch point). Segments are created by integrating outside a tetrahedron between the exit points, split points, and switch points. If a had a forward trace inside the current tetrahedron before we started propagation, this trace and a will be removed from the list. (We can do this after we have re-calculated this trace and replaced the new exit point by the one already in the front.) b is handled in the same manner.

4.3 Special Case Treatment

In the previous sub-section, we have made three assumptions:

(A1) Each stream line leaves the tetrahedron.

(A2) The stream surface can be triangulated by the end points, exit points, and split points.

(A3) The front does not split into isolated pieces in the tetrahedron.

The presence of critical points inside a tetrahedron can make any of these assumptions invalid. For example, a sink attracts stream lines; thus, part of the segment may end in the tetrahedron. We calculate the part of the segment that enters the critical point and propagate the remaining parts. This causes the front to split into two parts. The same observation holds for a saddle point, where a finite number of stream lines (often one) in the surface enters the critical point. In the absence of critical points, a stream line leaves the tetrahedron, as does the whole stream surface defined by a line segment.

4.4 Termination Conditions

We have discussed situations where parts of a segment or a whole segment end at a critical point. The typical case is that a segment ends at a boundary face. In our representation, we remove the segment from the front and split the front into two lists. A list is removed when no segment is left. Unfortunately, stopping only at critical points and the boundary may lead to infinite loops of propagating segments. It is possible that a stream surface "rolls" around a closed stream line or torus forever, see [1], [3]. Besides, it is desirable to provide an upper bound on run time. We accomplish this by limiting the number of intersections of a stream surface with any triangular face to one in each direction. Other conditions are possible, and reducing the number of resulting triangles is an obvious topic for further research.

4.5 Merging

Sub-section 4.2 implies that each split increases the number of segments, so do the switch points. Also, the exit lines consist of several segments resulting in an incrase of the number of segments. There is no step so far that removes segments (only traces) except the termination conditions. Therefore, the number of segments increases except when reaching the boundary or a critical point. For this reason, we always try to remove intermediate points on the exit curves. If the error induced by replacing (a^{n-2}, a^{n-1}, a^n) by (a^{n-2}, a^n) is smaller than our error bound ϵ, we remove a^{n-1} from the exit curve. Of course, we can never remove exit points, switch points and split points, since they are end points of exit curves on faces.

5 Results

We have tested our algorithm on a simple grid and random vector field to detect problems by creating strong twists in the surface. The grid contains $4 \times 4 \times 20$ points on a regular grid split into 1026 tetrahedra. A stream surface is shown in Figure 9. The surface shows twists and exhibits convergent regions. This type of vector field causes no problems for the algorithm since it concentrates on one tetrahedron at a time and we can obtain all information about the surface in this tetrahedron. We have included two streamlines calculated with an adaptive Runge-Kutta-Fehlberg method to illustrate the expected relationship between surface and streamlines.

Our second test used the well known bluntfin data set, courtesy of NASA. The stream surfaces shown in Figures 10 and 11 are all started at the boundary of the grid by performing integration against the flow direction into the interesting region around the fin. One can clearly see the flat, uninteresting area in the back as well as

the winding and twisting close to the fin. We hope to analyze the surfaces further in the future to obtain structural information.

It is not difficult to see that our algorithm is slower than Hultquist's method if precision is not much of an issue. Hultquist's method can give nice looking surfaces using only about one single point evaluation per cell. Since we need the exit, switch and split points, we have to calculate more points. In our experiments, we have had reasonable results with about 5 points per trace per tetrahedron. For a comparison of the precision of the two algorithms, we use a second small data set with random vector data. The data set does not contain critical points which are especially difficult for the Hultquist algorithm to deal with. The idea of the test is to calculate a stream surface generated by a single line segment, reverse the direction of flow and use the end of the first stream surface as starting segments for the second stream surface. In theory, one should get the original single line segment back. In Figure 12, we show the first surface for our algorithm. Then, the direction of flow is reversed and a second stream surface using the intersection of the shown surface with the boundary is used as start segment. For a better comparison, we presented only the final intersection of this second stream surface with the boundary and the original line segment in Figure 14. There are some numerical errors visible, especially around the center of the line segment. Now, we look at Hultquist's method. Figure 13 shows the surface generated by our single line segment. Again, we reverse the flow and use the intersection of the presented surface as input. In Figure 15, we show the result of the test in this case. It is clearly visible that the error is much larger. For an evaluation, it is important to note that we used a very small error bound for the adaptive integration, so that the Hultquist algorithm took four times longer than our algorithm to produce this result.

Acknowledgements

The work was supported by the Deutsche Forschungsgemeinschaft (DFG) under the program "Visualisierung nichtlinearer Vektorfeldtopologie," awarded to the University of Kaiserslautern, Germany. We thank all members of the Computer Graphics Group at the University of Kaiserslautern for many fruitful discussions.

This work was also supported by the National Science Foundation under contracts ACI 9624034 (CAREER Awards), through the Large Scientific and Software Data Set Visualization (LSSDSV) program under contract ACI 9982251, and through the National Partnership for Advanced Computational Infrastructure (NPACI); the Office of Naval Research under contract N00014-97-1-0222; the Army Research Office under contract ARO 36598-MA-RIP; the NASA Ames Research Center through an NRA award under contract NAG2-1216; the Lawrence Livermore National Laboratory under ASCI ASAP Level-2 Memorandum Agreement B347878 and under Memorandum Agreement B503159; and the North Atlantic Treaty Organization (NATO) under contract CRG.971628 awarded to the University of California, Davis. We also acknowledge the support of ALSTOM Schilling Robotics, and Silicon Graphics. We thank the members of the Visualization and Graphics Research Group at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis.

References

[1] R. H. Abraham and C. D. Shaw. *Dynamics, the Geometry of Behavior I–IV*. Aerial Press, Santa Cruz, CA, 1982, 1983, 1985, 1988.

[2] V. I. Arnold. *Ordinary Differential Equations*. Springer-Verlag, Berlin, 1992.

[3] J. Guckenheimer and P. Holmes. *Dynamical Systems and Bifurcation of Vector Fields*. Springer-Verlag, New York, NY, 1983.

[4] J. L. Helman and L. Hesselink. Visualizing Vector Field Topology in Fluid Flows. *IEEE Computer Graphics and Applications*, 11(3):36–46, May 1991.

[5] M. W. Hirsch and S. Smale. *Differential Equations, Dynamical Systems and Linear Algebra*. Academic Press, New York, NY, 1974.

[6] J. P. M. Hultquist. Constructing Stream Surfaces in Steady 3D Vector Fields. In A. E. Kaufman and G. M. Nielson, editors, *Proceedings of IEEE Visualization '92*, pages 171 – 178, Los Alamitos, CA, 1992. IEEE Computer Society.

[7] D. N. Kenwright and R. Haimes. Vortex Identification - Applications in Aerodynamics: A Case Study. In R. Yagel and H. Hagen, editors, *Proceedings of IEEE Visualization '97*, pages 413–416, Los Alamitos, CA, 1997. IEEE Computer Society Press.

[8] D. N. Kenwright and D. A. Lane. Optimization of Time-Dependent Particle Tracing Using Tetrahedral Decomposition. In G. M. Nielson and D. Silver, editors, *Proceedings of IEEE Visualization '95*, pages 321 – 328, Los Alamitos, CA, 1995. IEEE Computer Society Press.

[9] G. M. Nielson and I.-H. Jung. Tools for Computing Tangent Curves for Linearly Varying Vector Fields over Tetrahedral Domains. *IEEE Transactions for Visualization and Computer Graphics*, 5(4):360 – 372, 1999.

[10] R. Peikert and M. Roth. The "Parallel Vectors" Operator-a Vector Field Visualization Primitive. In *Proceedings of IEEE Visualization '99*, pages 263–270, Los Alamitos, CA, 1999. IEEE Computer Society.

[11] G. Scheuermann, B. Hamann, K. I. Joy, and W. Kollmann. Visualizing Local Vector Field Topology. *Journal of Electronic Imaging*, 9(4):356 – 367, 2000.

[12] M. Stämpfle. Dynamical Systems Flow Computation by Adaptive Triangulation Methods. *Computing and Visualization in Science*, 2(1):15–24, 1999.

[13] J. J. van Wijk. Implicit Stream Surfaces. In *Proceedings of IEEE Visualization '93*, pages 254 – 252, Los Alamitos, CA, 1993. IEEE Computer Society.

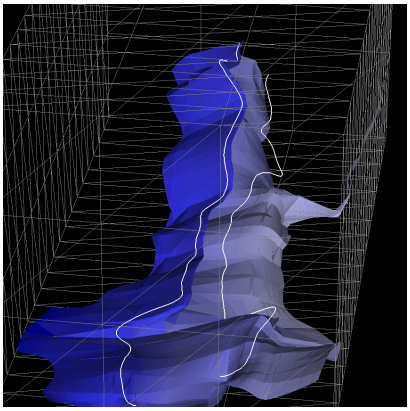

Figure 9: Stream surface calculated in random vector field. Random data creates a maximum on twisting, folding, and parts with converging and diverging flow. Example data set used for testing and debugging purposes.

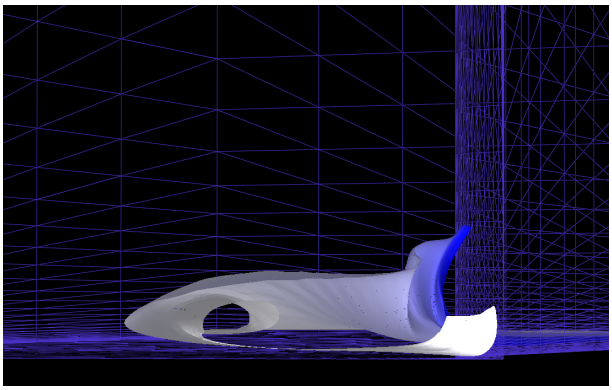

Figure 10: This stream surface in the bluntfin data set (courtesy of NASA) starts along a grid line at the rear. This surface exhibits "simple behavior" in the back, but it starts to wind and climb up the wall near the fin, indicating more complicated behavior.

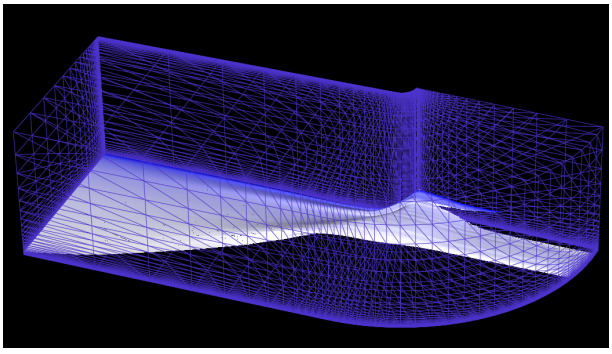

Figure 11: Stream surface in the bluntfin data set with a view on the whole grid.

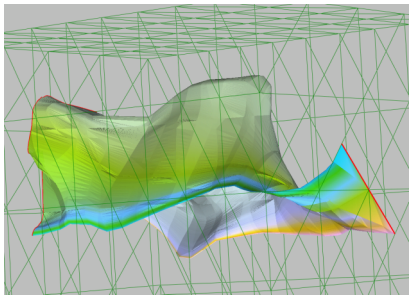

Figure 12: Stream surface from our algorithm in the precision test.

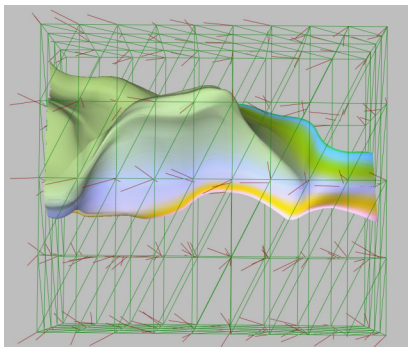

Figure 13: Stream surface from Hultquist's method in the precision test.

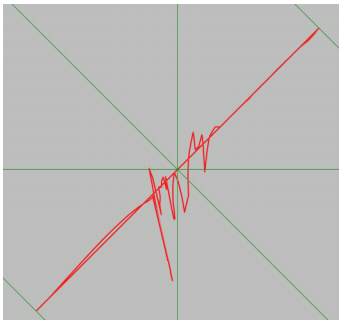

Figure 14: Our result in the precision test.

Figure 15: Hultquist's method in the precision test.

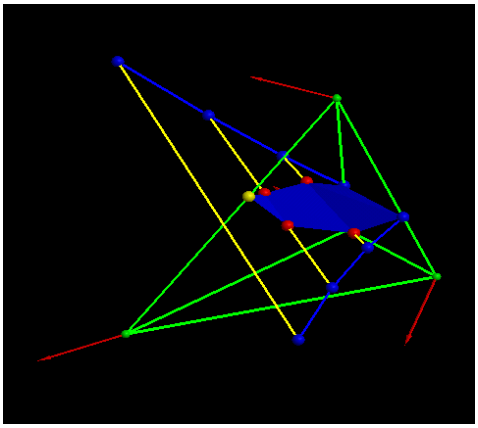

Figure 1: Stream surface construction inside tetrahedron. The endpoints are traced as streamlines. Each intermediate point is connected to its counterpart by a line on the surface. Clipping against the tetrahedron's faces yields the surface.

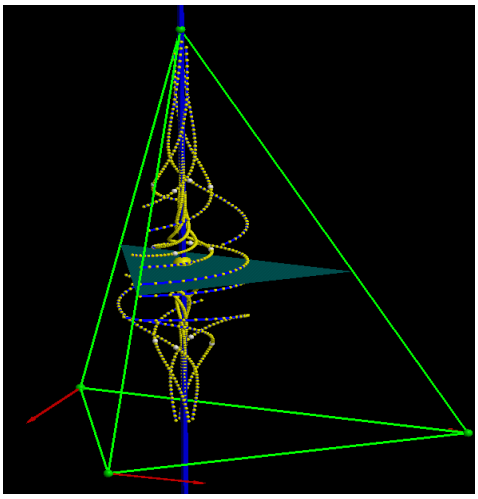

Figure 3: Vector field for case C2 exhibiting spiraling behavior around axis given by eigenspace of real eigenvalue. Critical point and rotation axis are stream lines themselves.

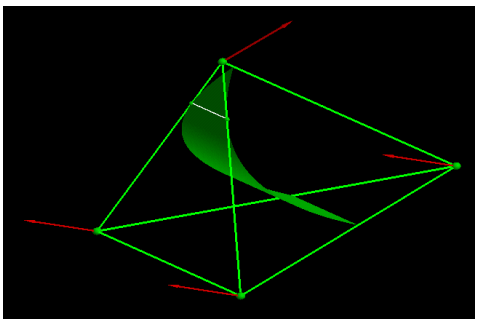

Figure 7: A switch line is defined as border between inflow and outflow on a tetrahedron face. As a result of linear interpolation, there can be only one switch line per face.

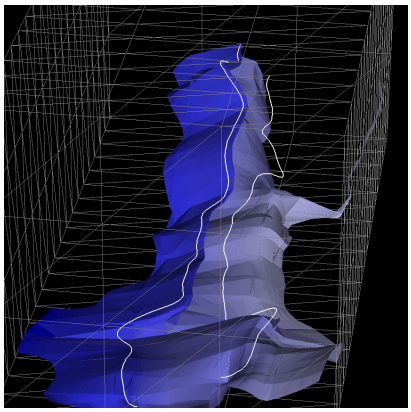

Figure 9: Stream surface calculated in random vector field. Random data creates a maximum on twisting, folding, and parts with converging and diverging flow. Example data set used for testing and debugging purposes.

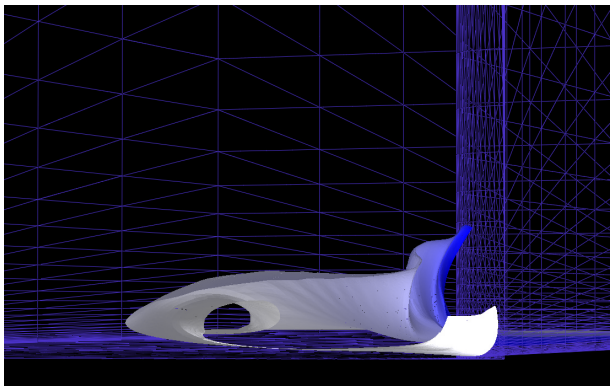

Figure 10: This stream surface in the bluntfin data set (courtesy of NASA) starts along a grid line at the rear. This surface exhibits "simple behavior" in the back, but it starts to wind and climb up the wall near the fin, indicating more complicated behavior.

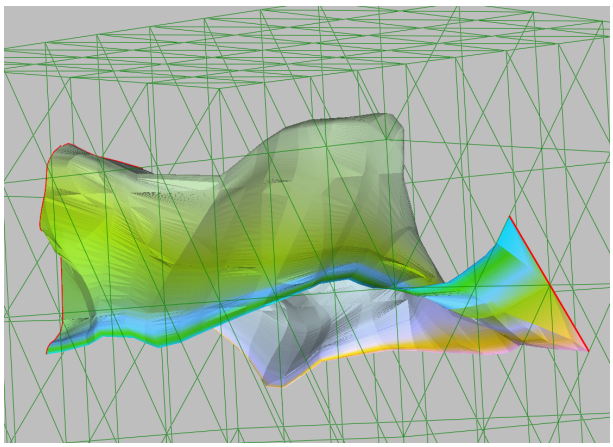

Figure 12: Stream surface from our algorithm in the precision test.