Stream Surface Generation for Fluid Flow Solutions on Curvilinear Grids

Allen Van Gelder

Computer Science Department
University of California, Santa Cruz, USA avg@cs.ucsc.edu

Abstract. A *stream surface* in a steady-state three-dimensional fluid flow vector field is a surface across which there is no flow. Stream surfaces can be useful for visualization because the amount of data presented in one visualization can be confined to a manageable quantity in a physically meaningful way.
This paper describes a method for generation of stream surfaces, given a three-dimensional vector field defined on a curvilinear grid. The method can be characterized as *semi-global*; that is, it tries to find a surface that satisfies constraints over a region, expressed as integrals (actually sums, due to discreteness), rather than locally propagating the solution of a differential equation.
The solution is formulated as a series of quadratic minimization problems in n variables, where n is the cross-wind resolution of the grid. An efficient solution method is developed that exploits the fact that the matrix of each quadratic form is tridiagonal and symmetric. Significant numerical issues are addressed, including degeneracies in the tridiagonal matrix and degeneracies in the grid, both of which are typical for the applications envisioned.

1 Introduction

Fluid flows include both gases and liquids, and are important in the design of many machines. Because of their complexity in the neighborhood of solid objects (even greatly simplified objects), almost all fluid flows are computed numerically on a grid or grids of some kind. The solutions involve both scalar and vector quantities, and can run to gigabytes of data. Significant simulations usually require super-computers.

Visualization of fluid flows is one of the most challenging visualization problems. Among the reasons for this are the fact that the grids on which the solutions have been computed are shaped to fit the boundaries of important solid objects. Some examples are aircraft fuselages, propellers, turbine blades, ship hulls, internal combustion cylinders, and ocean floors.

A second aspect that makes the visualization task challenging is that three-dimensional vector fields contain an overwhelming amount of information. One of our primary motivations for producing stream surfaces is to reduce the amount of information to a manageable, yet informative, quantity.

A third visualization challenge arises in many engineering applications because the flow needs to be studied in the close neighborhood of so-called *no-slip* boundaries: that is, boundaries of solid objects where the flow velocity is constrained to be zero

[KHL99]. So another motivation for stream surfaces is to produce a surface *near* the no-slip boundary such that the flow along this surface provides the needed insights.

In this paper we concentrate on *curvilinear grids*, whose regularity makes them attractive for computation. A curvilinear grid can be thought of as a continuous deformation of a rectangular parallelepiped, with *cells* that are warped cubes. Each vertex is identified by a triple index (i, j, k) and edges connect vertices that differ by one in exactly one index.

Methods to generate stream surfaces can be broadly grouped into two categories, *local* and *global*. The *local* methods generate stream lines from various points (usually on an upwind boundary face of the grid) by solving the differential equation implied by the vector field of the flow. Then somehow these stream lines need to be connected to make a surface. Substantial difficulties have been reported, including numerical inaccuracies that grow as the stream lines are propagated downwind.

Global methods attempt to overcome some of the drawbacks of local methods. The general idea of global methods is to choose points on the surface that are connected up as surface patches in such a way that the normal vectors of the patches are orthogonal to the flow field; error is measured as the degree of nonorthogonality. By applying the error same criterion at every patch, it is hoped to avoid large errors downwind due to accumulated inaccuracies from upwind. However, global methods have their own computation difficulties, and the hoped-for accuracy might not be achieved. In this paper we describe a limited approach, point out some of the problems that arose, and explain how we addressed those problems.

Fluid flows satisfy several conservation properties, one of the most important being conservation of mass: In a steady-state flow the net mass flowing across a closed surface is zero. There is no clear-cut way to exploit this property by locally propagating stream lines. However, it can be exploited in several ways with global methods, providing another motivation for a global approach.

The flow field can be transformed into parameter space (also called computational space) while preserving the mass-conservation property, provided that the vector field represents *momentum* (which is velocity weighted by density). Therefore a stream surface in the geometrically regular computational space can be mapped directly into physical space and it will remain a stream surface.

We shall demonstrate that, although this transformation into computational space may have a geometric singularity that causes the velocity to become undefined at certain points in computational space, the *computational space momentum* remains well-defined and finite. Such geometric singularities arise in practice when the longitudinal lines of a hemisphere converge to a single point at the pole. Such grid shapes are needed to fit around the nose of an aircraft fuselage and similarly shaped objects.

The paper is organized as follows. The problem we address is specified in Section 2. The methodology we developed is reported in Section 3. Experimental results are reported in Section 4. Related work is discussed briefly in Section 5, and conclusions are discussed in Section 6. Many details that we are forced to omit here due to space limitations, some additional images, and animations can be accessed on the Internet at `ftp://ftp.cse.ucsc.edu/pub/avg/Scivi`.

2 The Problem

The data given consists of a three-dimensional array of grid points, $p(i, j, k)$, also called grid vertices, and momentum vectors $\mathbf{m}(i, j, k)$, where $0 \leq i < m$ and $0 \leq j < n$ and $0 \leq k < K$. Each grid point has components $p = (x, y, z)$. Each momentum vector has components $\mathbf{m} = (u, v, w)$, and $\mathbf{m}(i, j, k)$ denotes the flow at $p(i, j, k)$. Also part of the data are scalar fields for density, $\rho(i, j, k)$, and stagnation energy, $E(i, j, k)$, but E will not play a direct role in this paper. Notice that (u, v, w) denote momentum, not velocity, for this paper; the relationship is $(u, v, w) = \rho(\dot{x}, \dot{y}, \dot{z})$, where $(\dot{x}, \dot{y}, \dot{z})$ is velocity.

We assume that the flow is generally in the direction of $\partial p/\partial x$, in accordance with the convention of fluid mechanics. We also assume that the surface $p(i, j, 0)$ is (at least in part) a no-slip boundary, where $\mathbf{m}$ is zero. The latter assumption is not required, but no-slip boundaries present important difficulties, so we include it in the problem.

The upwind face of the grid is comprised of the vertices $p(0, j, k)$. The problem we study is to find a stream surface in the flow whose intersection with this upwind face is specified. That is, suppose a one-dimensional family of "heights" is given as $\psi(0, j)$. Associate a point $p(0, j, \psi(0, j))$ with each given "height" by interpolation among the given points $p(0, j, k)$. Then we look for the stream surface that passes through the points $p(0, j, \psi(0, j))$. For this paper, trilinear interpolation is used generally, so that if k is the integer such that $k \leq \psi(0, j) < k + 1$, then the intersection point computed by linear interpolation along the edge between $p(0, j, k)$ and $p(0, j, k + 1)$.

Grids arise frequently in which the points near $i = 0$ approximate a spherical or ellipsoidal cap. In this case, for a fixed k, the points $p(0, j, k)$ are all in the same location for varying j, as shown in Figure 1. Grid cells adjacent to $i = 0$ become wedges. A stream surface can be continuous in this region only if the "heights" $\psi(0, j)$ are all equal. This is one motivation for specifying the intersection of the desired stream surface with the upwind face.

Actually, we expect to want a series of stream surfaces that are near the no-slip surface and nonintersecting. They should be varying "distances" from the no-slip surface. By adjusting the "heights" at the upwind face, surfaces at various distances can be generated. If the surfaces do not intersect at the upwind face, they will not intersect anywhere, in theory (or mass would be destroyed).

3 Methodology

The general approach we followed was to define the conditions that the dot product of the surface normal with the flow vector field should be zero, or close to zero. When the problem is viewed this way, it can be transformed into computational space. This section first describes the transformation into computational space, and how certain degeneracies are handled. Then it gives the constraints to be satisfied by the stream surface. Finally, it describes the methods we implemented to satisfy those constraints.

3.1 Transformation into Computational Space

Suppose a rectilinear region in parameter space (r, s, q), is considered, where $0 \leq r \leq m-1$, $0 \leq s \leq n-1$, $0 \leq q \leq K-1$ are continuous parameters. Trilinear transformations can be defined in each unit cube with integer boundaries for each of the quantities, x, y, z, u, v, w, as well as ρ and E. Consider a fixed triple of integers (i, j, k), such that the cell is the region $i \leq r \leq i+1$, $j \leq s \leq j+1$, $k \leq q \leq k+1$.

To make the presentation independent of (i, j, k) we define a local coordinate system (α, β, γ) for each cell, where the local coordinates vary from 0 to 1.

Notation: Partial derivatives with respect to α, β, and γ are denoted by subscripting: $p_\alpha = \partial p / \partial \alpha$, etc. Vectors are considered to be column vectors; the superscript T denotes *transpose*, converting a column vector into a row vector. A matrix is sometimes denoted by a row of column vectors or a column of row vectors. The usual 3D cross product (vector product) is denoted by "$\times$" and the usual dot product (inner product) is denoted by "$\cdot$".

Let T be the 3-vector of trilinear transformations that maps $(0,0,0)$ into $p(i,j,k)$, $(1,0,0)$ into $p(i+1,j,k)$, $(0,1,0)$ into $p(i,j+1,k)$, $(0,0,1)$ into $p(i,j,k+1)$, etc. We write $p(\alpha,\beta,\gamma) = T(\alpha,\beta,\gamma)$ within this cell. The Jacobian matrix of T is

$$J = [p_\alpha \;\; p_\beta \;\; p_\gamma] \tag{1}$$

where p is regarded as a column vector. Let $\|J\|$ denote the determinant of J, which we assume is nonnegative. Both J and $\|J\|$ vary with (α,β,γ), but this dependence is suppressed in the notation.

For trilinear transformations, recall that the partial derivatives that appear in Eq. 1 are simple vector differences at the cell corners. For example,

$$p_\alpha(0,0,0) = p_\alpha(1,0,0) \;=\; p(i+1,j,k) - p(i,j,k) \tag{2}$$

and so on. Elsewhere in the cell p_α is a bilinear function of β and γ.

We want to define computational-space analogs of the basic quantities given in the data: momentum, density and energy. With some abuse of notation we indicate computational-space quantities with the same symbols and their physical-space counterparts, and use the parameters (α,β,γ) or (x,y,z) to distinguish which is intended. Details of the derivations are omitted due to lack of space.

It is known, at least in the folklore of CFD, that *computational-space density* is given by

$$\rho(\alpha,\beta,\gamma) = \|J\|\,\rho(x,y,z). \tag{3}$$

However, the following expression for *computational-space momentum* has not appeared elsewhere to the best of our knowledge. It relies on the relationship between cross-products and the inverse of a 3x3 matrix, which is not "well advertised."

$$\mathbf{m}(\alpha,\beta,\gamma) = \begin{bmatrix} (p_\beta \times p_\gamma)^T \\ (p_\gamma \times p_\alpha)^T \\ (p_\alpha \times p_\beta)^T \end{bmatrix} \mathbf{m}(x,y,z) \tag{4}$$

A very important property of this definition is that the mass-conservation law holds in computational space, as well as in physical space. If we define a stream surface in computational space to be a surface whose normal vector is everywhere orthogonal to $\mathbf{m}(\alpha, \beta, \gamma)$, then the mapping of this surface into physical space is also a stream surface, since no mass crosses either surface.

We also have an expression for the determinant in terms of the triple scalar product of the columns of the matrix:

$$\|J\| = (p_\alpha \times p_\beta) \cdot p_\gamma \tag{5}$$

As mentioned in connection with Eq. 1, for trilinear transformations, Eqs. 4 and 5 can be evaluated at cell corners with simple vector differences, cross products, and dot products.

3.2 Degeneracies in the Jacobian Matrix

It is normal for the Jacobian matrix to have singularities in common modeling situations. A typical case is when the grid face for $i = 0$ collapses into a line (or polygonal line) as edges shrink to 0 length in the j direction. The effect is like the end of a cylinder being pinched down to a single point. Therefore, it is quite useful to have a representation in computational space that does not depend on the transformation being 1–1, that is, does not depend on the Jacobian matrix having an inverse. Equations 3 and 4 described such a representation.

3.3 The General Scheme

We construct stream surfaces in computational space on the assumption that they should not intersect the $q = 0$ face of the grid. The stream surface will be defined as a set of quadrilateral patches in (r, s, q) space (called (i, j, k) space at integers), where the vertices of the quadrilaterals have integer values for r and s, and the values of q can be thought of as a height field, $\psi(r, s)$, over the $q = 0$ plane. The edges of these patches run from $(i, j, \psi(i, j))$ to $(i+1, j, \psi(i+1, j))$ or from $(i, j, \psi(i, j))$ to $(i, j+1, \psi(i, j+1))$. See Figure 2.

Clearly this scheme is not sufficiently general to generate any well-defined stream surface. However, we are primarily interested in the case that $q = 0$, or $k = 0$, is the location of a no-slip surface. The region near such a surface is called the *boundary layer* and flows within the boundary layer are often the most important for design purposes, because they determine the impact of the environment on the machine being designed. This representation should be adequate when the flow is “somewhat parallel” to the no-slip surface.

However, there are important exceptions where the representation as a height field is not adequate. These occur, for example, where the flow becomes generally toward the no-slip surface (an *attachment* flow) or generally away from the no-slip surface (a *separation* flow). Normally, the u component of computational-space momentum is negative somewhere near such regions. Our basic method of generating a stream surface breaks down in such regions, and some alternative is needed, as discussed in Section 3.6.

In general terms, our goal is to choose a height field $\psi(i, j)$ for $0 \leq i < m$ and $0 \leq j < n$ such that the mass under this height field is a specified amount, and the flux (flow normal to the surface defined by the height field) is as close to zero as possible, in some sense. In addition, the upwind edge of the surface, $\psi(0,j)$ must satisfy some further shape constraint to make the solution unique; the shape constraint we have used is that $\psi(0,j)$ must be constant.

The method that we have implemented extends the surface in strips. Each new strip runs across j for the next higher value of i. Thus the first strip defines values for $\psi(1,j)$ and once they are defined, the second strip defines values for $\psi(2,j)$, and so on (see Figure 2).

Without going into detail on the mathematics, the i-th strip is generated as follows (values of $\psi(i-1,j)$ have already been decided):

1. Choose provisional values for $\psi(i,j)$; for example, $\psi(i,j) = \psi(i-1,j)$ to start with.
2. In each quadrilateral patch running from $i-1$ to i and from $j-1$ to j, compute the coefficients of a bilinear model of $\mathbf{m}(\alpha,\beta)$.
3. In each quadrilateral patch as above, express the normal vector throughout the patch, treating $\psi(i,j-1)$ and $\psi(i,j)$ as variables.
4. In each quadrilateral patch as above, compute the integral of the squared flux, still treating $\psi(i,j-1)$ and $\psi(i,j)$ as variables, but using the bilinear model of $\mathbf{m}(\alpha,\beta)$. The flux is the dot product of the momentum and the normal vector. The integral can be found in closed form and is a quadratic expression in the variables $\psi(i,j-1)$ and $\psi(i,j)$.
5. The sum of all such quadratic expressions over the whole strip is the squared flux to be minimized. Define x_j to be the n-vector of values of $(\psi(i,j) - \psi(i-1,j))$, for $0 \le j < n$, that minimizes the quadratic form. The x_j's can be found by solving the linear system that represents the gradient of the quadratic form. Because x_j is only involved in constraints with x_{j-1} and x_{j+1}, the matrix of the linear system is tridiagonal. Because the linear system is the gradient of a quadratic form the matrix is symmetric.

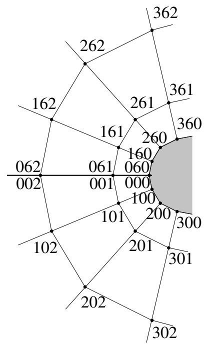

Fig. 1. To fit a curvilinear grid over a spherical cap, the $i = 0$ grid face collapses to the heavy line. Indexes are shown as ijk. For this example j varies from 0 to 12. The no-slip boundary is at $k = 0$.

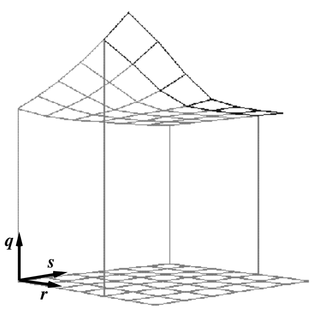

Fig. 2. Patches that make up the stream surface in computational space are shown above the $q = 0$ plane, with the newest strip on the right.

6. Use the solution x_j to compute new provisional values $\psi(i,j) = \psi(i-1,j) + x_j$. Compute an updated bilinear model of $\mathbf{m}(\alpha,\beta)$. Repeat the process until the squared flux is no longer decreasing or the solution stabilizes within a specified tolerance.

After all all the strips have been generated according to the above outline, compute the total mass under the surface and compare it with the amount that was required. Adjust the height of the upwind edge up or down until the required total mass is under the surface, within a specified tolerance.

Thus the procedure involves nested iterations. The inner iteration reduces the squared flux, and the outer iteration adjusts the overall surface height. The only saving grace is that the tridiagonal system can be solved very efficiently, in time that is linear in n. A routine matrix inversion would be order n^3.

3.4 Details of the Quadratic Form

This section sketches the development of the quadratic form that is minimized. The basis is the analysis of one patch, i.e., one quadrilateral, whose vertices are at $(i-1, j+1, \psi(i-1,j+1))$, $(i-1, j, \psi(i-1,j))$, $(i, j+1, \psi(i,j+1))$, and $(i, j, \psi(i,j))$. Switching to the local coordinate system, the surface is modeled as the bilinear function $\psi(\alpha,\beta)$. The squared flux for this patch is:

$$F_j^2 = \int_0^1 \int_0^1 \left(-u(\alpha,\beta)\,\psi_\alpha - v(\alpha,\beta)\,\psi_\beta + w(\alpha,\beta)\right)^2 \, d\alpha\, d\beta \tag{6}$$

The integral has a closed form, in which the heights $\psi(i,j+1)$ and $\psi(i,j)$ appear quadratically. Let $x_j = (\psi(i,j) - \psi(i-1,j)$. The quadratic form to be minimized is

$$F^2 = \sum_{j=0}^{n-2} F_j^2 \tag{7}$$

which is nonnegative definite, so a minimum must exist.

The program contains an option to scale the squared flux inversely by the squared momentum; intuitively this makes the sine of the angle between the momentum and the surface patch the error, rather than the flux itself.

We define a series of terms leading to a tridiagonal linear system whose solution minimizes F^2. Momentum terms always refer to computational-space momentum and are values at the center of the patch (which are also averages over the patch). Subscripts of α and β denote partial derivatives (of the appropriate bilinear function). We introduce local symbols g, G, h, q, Q, r, R, s, S in the equations below; their purpose is to obtain expressions for a_j, b_j, and c_j.

$$\begin{aligned}
g_j &= (-u+v)/2 + (u_\beta + v_\alpha)/12 & G_j &= (-u-v)/2 - (u_\beta + v_\alpha)/12 \\
q_j &= -u + (u_\beta - v_\beta)/2 & Q_j &= -v + (u_\alpha - v_\alpha)/2 \\
r_j &= u + (u_\beta + v_\beta)/2 & R_j &= v + (u_\alpha + v_\alpha)/2
\end{aligned}$$

$$a_j = g_j^2 + (q_j^2 + Q_j^2)/12 + G_{j-1}^2 + (r_{j-1}^2 + R_{j-1}^2)/12 \tag{8}$$

$$b_j = G_j g_j + (r_j q_j + R_j Q_j)/12 \tag{9}$$

$$h_j = v(\psi(i-1,j+1) - \psi(i-1,j)) - w$$
$$s_j = -v_\beta(\psi(i-1,j+1) - \psi(i-1,j)) + w_\beta$$
$$S_j = -v_\alpha(\psi(i-1,j+1) - \psi(i-1,j)) + w_\alpha$$
$$c_j = h_j g_j + h_{j-1} G_{j-1} + (s_j q_j + S_j Q_j) + s_{j-1} r_{j-1} + S_{j-1} R_{j-1})/12 \quad (10)$$

Any terms containing subscripts outside the range 0 through $n-1$ are considered 0.

A linear system is defined using Eqs. 8–10. The vector c has c_j as its components. Finally, we define the matrix A whose entries are 0, except for three diagonals:

$$A_{jj} = a_j \qquad A_{j,j+1} = A_{j+1,j} = b_j. \quad (11)$$

The linear system to be solved is $Ax = c$.

3.5 Solving the Tridiagonal System

The results of this section are obtained with standard methods of linear algebra and vector calculus, which may be found in many college texts on the subject. Let $A = L\,L^T$ be the Cholsky factorization of A. That is, L is lower triangular. Then L has the form $L_{jj} = d_j$, $L_{j+1,j} = e_j$, and all other entries are 0. With the convention that $e_{-1} = 0$, the d_j and e_j are defined by the recurrence relations:

$$d_j^2 = a_j - e_{j-1}^2 \quad (12)$$
$$e_j = b_j / d_j \quad (13)$$

Some care is required to avoid numerical problems. Equations 8 and 10 allow us to infer special properties of linear system. The key is to interpret them in terms of inner products of certain combinations of the vectors: (g_j, q_j, Q_j), $(G_j r_j, R_j)$, and (h_j, s_j, S_j), $0 \le j < n$. It can be shown that the right-hand side of Eq. 12 must be nonnegative, so if a negative value is obtained, it is due to numerical inaccuracies and may be replaced by 0. Also, it can be shown that if $d_j = 0$ in Eq. 13, then b_j must also be 0, and the equations of rows 0 through j separate from (have no variables in common with) those in rows $j+1$ through $n-1$. This independence can be implemented by setting e_j to 0, rather than using Eq. 13. Similarly, if any $a_j = 0$, then Eqs. 8 and 9 imply that b_{j-1} and b_j are 0, and a similar separation occurs.

To solve $Ax = c$, we compute the intermediate result $f = L^{-1}c$, followed by $x = (L^T)^{-1} f$. The procedure uses a forward recurrence equation, followed by a backward recurrence equation. As usual, f_{-1} and x_n are considered to be 0.

$$f_j = (c_j - e_{j-1} f_{j-1}) / d_j \quad (14)$$
$$x_j = (f_j - e_j x_{j+1}) / d_j \quad (15)$$

Again, due to separations mentioned above, whenever $d_j = 0$, then f_j and x_j can be set to 0, rather than using the recurrence equations. This rule delivers a valid solution of $Ax = c$, although the solution is not unique because A is singular.

3.6 Additional Numerical Issues

If the w component of the computational-space momentum dominates the u component, then the inner iteration described in Section 3.3 does not converge, for essentially the same reasons that a nonadaptive Runge-Kutta procedure does not converge when the derivative is too large. Because it is common for cells to be very thin in the k direction near a no-slip boundary, even a vector that is nearly parallel to the no-slip surface in physical space can become nearly orthogonal to the no-slip surface in computational space. We address this problem in two ways.

1. If the solution vector x does not lead to a smaller squared flux than the previous provisional value, then some fraction of x between 0 and 1 is searched for that *does* reduce the squared flux. If no satisfactory fraction is found, the iteration terminates with the previous provisional value becoming the final value.
2. If the values of ψ in question are less than 1, where 0 is the location of a no-slip surface, then we interpolate $w(i,j,\psi) = \psi^2\, w(i,j,1)$, rather than linearly. There is physical justification for this: Boundary layer theory going back to the classical work of Blasius shows that w varies quadratically in the neighborhood of the no-slip boundary (while u remains linear). This method of interpolation allows the interpolated momentum vector to approach tangency with the no-slip surface, as is required by conservation of mass.

The moral is that linearizing an inherently nonlinear system does not guarantee good results.

If the u component of the computational-space momentum changes sign between $i-1$ and i then there might be an asymptotic surface that passes between $(i-1, j, \psi(i-1, j))$ and $(i, j, \psi(i, j))$. That is, the surface we have propagated to $i-1$ approaches but does not cross this asymptotic surface. A rigorous treatment of this situation would involve an eigenvalue and eigenvector analysis of the linearized 3D vector field in the neighborhood, which is beyond the scope of this paper. The main idea is to identify the two eigenvalues whose real parts have the same sign (all three cannot agree on sign or mass is not conserved). Then the plane that "corresponds to" these two eigenvalues is the asymptotic plane.

For now we have implemented a stop-gap solution whenever u is negative. We assume that u will be positive if we get far enough away from the no-slip boundary. So if $u(i, j, \psi(i, j)) < 0$ for some value of ψ that is being considered, we increase $\psi(i, j)$ until it is positive, or leaves the grid. If it leaves the grid, then we assume that u is a positive "free stream" value there, and compute accordingly.

4 Experimental Results

We have implemented programs to convert CFD solutions into computational space and to generate stream surfaces. We used the NASA/NAS FAST program to produce images. This section reports some experimental results. Timing results are all based on an SGI Onyx 2 using one 195 MHz R10000 processor. Solution quality is measured by the root-mean-square (RMS) sine of the angle between the momentum vector and the tangent plane of the surface. Figures cited in this section appear in the color plates.

We omit detailed results on a 9x9x9 test dataset with a quadratic vector field, mass-conservative flow, and no-slip surface. It confirmed that the program converges quickly on "nice" data.

The first dataset we report on is a simulation of the space shuttle launch vehicle during ascent. The simulation used 9 zones in its grid, but we analyzed only the third zone, and further limited that to the part having a no-slip boundary. This comprised an 80x77x48 grid, 295,680 vertices in all. We generated 15 stream surfaces in 799 CPU seconds. The RMS error varied in a narrow range from 0.286 to 0.304.

Several images are shown in Figure 3. Part (a) shows the no-slip fuselage for reference. Part (b) shows the third stream surface in a family of 15 in purple. Notice that the surface needs to begin well in front of the nose in order to be tangent to the flow. The grid degeneracies in front of the nose did not cause any computational difficulties. Part (c) shows the stream surface separating from the underneath of the fuselage.

Part (d) is another view. Notice the "ear" that appears where we might expect a tail fin (but there is no tail fin). This is a region of turbulent flow due to the bulge of the engine housing. The lowest part of the stream surface (furthest from the fuselage) is in a region where the computational-space value of u is negative, and the stop-gap method mentioned in Section 3.6 came into play. Although the stream surface might be inaccurate in these regions, it indicates visually that "something is happening," so other tools can be used to investigate more closely.

The final dataset we report on is a steady-state delta wing simulation with 15-degree angle of attack. The entire grid is 67x209x49 but again we limited it to the portion over a no-slip boundary, the size of which was 47x209x49, or 481,327 vertices. We generated 3 stream surfaces in 2725 CPU seconds. The RMS error varied from 0.320 to 0.336.

Figure 4 shows the third stream surface, counting from the no-slip "fuselage." Part (e) shows an overview from the rear and above. Notice where the stream surface separates from the "wing" surface near the outer edges forming a few channels. Part (f) shows a closer view of one side. These generally parallel lines of separation and attachment confirm and supplement the observations of "open" separation lines by Kenwright, Henze, and Levit [KHL99].

5 Related Work

Lack of space prevents an extensive review of the literature; please see cited works for additional bibliography [Ken93,vW93,KM96]. Hultquist described the construction of stream surfaces by patching together *stream ribbons*, which in turn are produced by generating stream lines [Hul92].

Three-dimensional stream functions, also known as *Clebsch potentials*, are scalar functions whose isosurfaces are stream surfaces *Clebsch potentials* [Lam32]. They have been studied computationally by Kenwright and Mallinson [KM92], by Kenwright [Ken93], by van Wijk [vW93], by Knight and Mallinson [KM96], by Feng *et al.* [FWJ98], and others. The method of Knight and Mallinson is global, but it applies to tetrahedral grids and solenoidal (curl-free) flows. The method of van Wijk is also global, but (as reported) it is limited to incompressible flows, is apparently implemented only for regular grids, and only handles certain flow topologies. This method propagates stream lines backward through the velocity field, so it is unclear how it would handle no-slip surfaces, where the velocity is 0.

In general, the methods reported in the literature do not address the special problems in CFD flow data that our program tries to address, such as no-slip surfaces, grids with degeneracies, and somewhat noisy data that results from numerical PDE solutions. They often have restrictions that prevent them from operating on the data at all, so a detailed comparison is not practical.

6 Conclusion

We have presented a new method for generating stream surfaces. It is a "semi-global" method in the sense that it simultaneously solves constraints over a large region of space, rather than working in one local region at a time, yet there is an element of downstream propagation. Its efficiency depends on a fast procedure for solving tridiagonal linear systems. The implementation so far has limited flexibility. Work in progress addresses situations in which u is negative, and will be the subject of a future report. Future work should also deal with multiple no-slip surfaces.

Stream surfaces are best used in combination with other cues for visualization. In this paper they were usually supplemented with arrows at various points in the surface. The surface gives shape information and the arrows give direction within the surface and magnitude information.

Acknowledgments This work was supported in part by NASA-Ames grant NAG2-1239 and NSF grant CCR-9503829. The datasets are from the NASA/NAS group at Ames, except the test. We thank the Zentrum für Angewandte Informatik Köln within the Regionales Rechenzentrum (Regional Computing Center) of the University of Cologne for making their extensive computer resources available during the author's visit.

References

[FWJ98] D. Feng, C. Wenli, and S. Jiaoying. Stream surface construction using mass conservative interpolation. *J. Computer Science and Technology*, 13 (suppl.issue):45–53, 1998.

[Hul92] J. P. M. Hultquist. Constructing stream surfaces in steady 3D vector fields. In *Proceedings of Visualization '92*, pages 171–178, Boston, MA, October 1992. IEEE.

[Ken93] D. N. Kenwright. *Dual Stream Function Methods for Generating Three-Dimensional Streamlines*. PhD thesis, Department of Mechanical Engineering, University of Aukland, New Zealand, August 1993.

[KHL99] D. N. Kenwright, C. Henze, and C. Levit. Feature extraction of separation and attachment lines. *IEEE Trans. Visualization and Computer Graphics*, 5(2):135–144, 1999.

[KM92] D. N. Kenwright and G. D. Mallinson. A 3-D streamline tracking algorithm using dual stream functions. In *Visualization '92*, pages 62–68. IEEE, October 1992.

[KM96] D. Knight and G. D. Mallinson. Visualizing unstructured flow data using dual stream functions. *IEEE Trans. Visualization and Computer Graphics*, 2(4):355–363, 1996.

[Lam32] Horace Lamb. *Hydrodynamics*. Dover, 6th edition, 1932.

[vW93] J. J. van Wijk. Implicit stream surfaces. In *Visualization '93*, pages 245–252, San Jose, CA, 1993. IEEE Comput. Soc. Press.

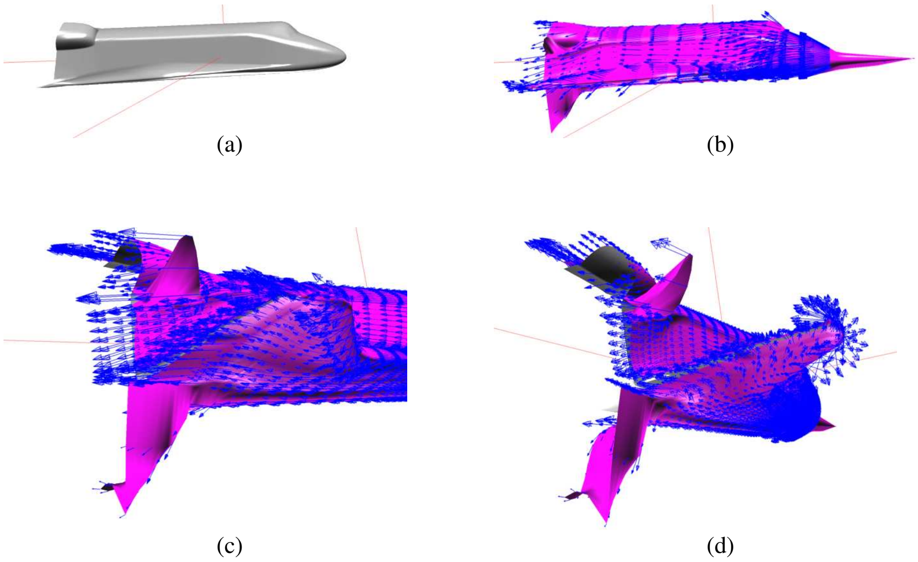

Fig. 3. Stream surface for simulation of the space shuttle launch vehicle, zone 3 of a nine-zone grid; see text for discussion.

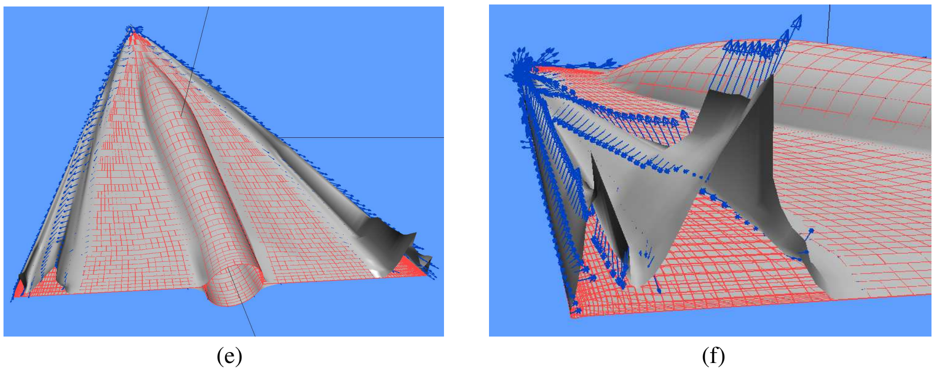

Fig. 4. Stream surface (smooth-shaded) in steady-state delta wing simulation, showing lines of separation and attachment near the outer edges. (e) Overview. (f) Closeup.