Efficient Streamline, Streamribbon, and Streamtube Constructions on Unstructured Grids

Shyh-Kuang Ueng, Christopher Sikorski, and Kwan-Liu Ma, *Member, IEEE*

Abstract—Streamline construction is one of the most fundamental techniques for visualizing steady flow fields. Streamribbons and streamtubes are extensions for visualizing the rotation and the expansion of the flow. This paper presents efficient algorithms for constructing streamlines, streamribbons, and streamtubes on unstructured grids. A specialized Runge-Kutta method is developed to speed up the tracing of streamlines. Explicit solutions are derived for calculating the angular rotation rates of streamribbons and the radii of streamtubes. In order to simplify mathematical formulations and reduce computational costs, all calculations are carried out in the canonical coordinate system instead of the physical coordinate system. The resulting speed-up in overall performance helps explore large flow fields.

1 Introduction

Streamlines are paths of massless particles which are released in a steady flow. Plotting of the particle paths produces a streamline picture which allows engineers to visualize fluid motion and to locate the regions of high and low velocity and, from these, the zones of high and low pressure. Additional information about the flow field, such as local flow rotation and expansion, can be shown in the form of a streamribbon and a streamtube.

Given a steady flow with velocity field $\vec{u}(\vec{x}(t))$, a streamline can be calculated by solving the following ordinary differential equation:

$$\frac{d\vec{x}(t)}{dt} = \vec{u}\big(\vec{x}(t)\big), \tag{1}$$

where t is the integration variable and is not to be confused with time [1].

The path swept by a deformable line segment becomes a streamribbon. Thus a streamribbon can reveal the translation, the angular rotation, and the rate of shear deformation of the flow. Volpe [2] creates a streamribbon by tracing a large number of adjacent streamlines. However, the number of streamlines needed to form smooth ribbon surfaces could be tremendous and the corresponding computational cost would be high. Therefore, in practice, the construction is simplified, though some information such as shear deformation would be lost.

In [3], a stream-surface, which is similar to a streamribbon, is generated by computing only a few streamlines and creating polygons between adjacent streamlines to form the surface of the stream-surface. This method requires complicated algorithms to deal with convergence, divergence and splitting of a stream-surface. Darmofal and Haimes [4], Ma and Smith [5], and Pagendarm [6] use one streamline and vectors normal to the local velocity to form a streamribbon. In this way, the resulting ribbons only represent the translation and the angular rotation of the flow. We adopt their algorithms by using two parallel edges to form a streamribbon.

Formally, a streamtube is defined as the surface formed by all streamlines passing through a given closed curve in the flow [1]. Streamtubes are used to visualize expansion, contraction and deformation of the flow. In [4], a streamtube is created by connecting the circular crossflow sections along a streamline. The radius of a cross flow section is determined by the local cross flow expansion rate. The streamtube constructed in this manner only reveals the flow expansion rate along the streamline. Since this technique is more computationally feasible, we adopt it in this work. In order not to be confused with the formal definition of the streamtube concept, hereafter, we use the name *iconic streamtube*.

In [5], to visualize both flow convection and diffusion, statistical dispersion of the fluid elements about a streamline is computed by using added scalar information about the mean square root value of the vector field and its Lagrangian time scale. The result defines the radius of the cross flow section and also forms a tube-like surface. In [7], Schroeder et al. describe a technique called stream polygon for visualizing local deformation and strain of the flow. In their work, local flow field information is represented by using a regular polygon which is perpendicular to the local vector field. A streamtube can be generated by sweeping the polygon along a streamline. The radius of the streamtube varies with the velocity magnitude, such that the mass flow is a constant along the streamtube.

In this paper, we describe new computational methods for fast streamline, streamribbon and iconic streamtube

- *S.-K. Ueng and C. Sikorski are with the Department of Computer Science, University of Utah, Salt Lake City, UT 84112. E-mail: {skueng, sikorski}@cs.utah.edu.*
- *K.-L. Ma is with the Institute for Computer Applications in Science and Engineering (ICASE), Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23681. E-mail: kma@icase.edu.*

For information on obtaining reprints of this article, please send e-mail to: transvcg@computer.org, and reference IEEECS Log Number V96016.

1077-2626/96$05.00 ©1996 IEEE

construction on unstructured grids. We assume that all input data cells are linear tetrahedra, which allows us to simplify some of the formulations. Data cells of other types must be decomposed into tetrahedra in a preprocessing step. After the decomposition, the original vector field is considered to be a piecewise linear vector field in the resulting mesh. A method is illustrated in [8] to subdivide hexahedral cells into tetrahedral cells.

In the finite element analysis, to achieve better computational efficiency, calculations are often done in the canonical coordinate system rather than in the physical coordinate system. As further explained in the next section, cells in the canonical coordinate system can be handled straightforwardly since they are normalized and transformed to the origin. In this work, we take the same approach. Test results show saving in both computational cost and memory requirements when compared to results from our previous research [9] which takes the the opposite approach. However, the results reported by Sadarjoen et al. [10] show otherwise. The main reason is that their test data are hexahedron cells instead of tetrahedron cells. For deformed hexahedra, the coefficients of the coordinate transformation function cannot be accurately calculated by using simple differencing methods. In order to improve accuracy, more sophisticated methods, thus more computationally expensive, must be used.

Our algorithms consist of the following steps:

- choose an initial point in the physical coordinate system.
- find the cell containing the initial point.
- transform the point to the canonical coordinate system.
- while the streamline construction is not completed, do:
  - * compute the next point along the particle path.
  - * for the streamribbon or the iconic streamtube construction, additional calculations are carried out.
  - * calculate and store the physical coordinates of the new results.
  - * locate the cell containing the point calculated in this step.

For generating particle paths, a specialized version of the fourth-order Runge-Kutta method (SRK4) is developed which requires only one matrix-vector multiplication and one vector-vector addition to calculate a new streamline point. The angular rotation rate of streamribbons and the radius of iconic streamtubes are governed by ordinary differential equations which were solved numerically in the past. Explicit solutions of these equations are now derived to speed up the construction of streamribbons and iconic streamtubes. Test results show our algorithms result in significant improvement in performance over traditional algorithms.

2 Coordinate Transformation

We assume that the edges of the cells are straight line segments. Therefore, a cell is a linear tetrahedron in both coordinate systems, and each canonical coordinate is regarded as a linear polynomial of the physical coordinates. The coordinate transformation function can be formulated as:

$$\vec{\xi} = T * \vec{x} + \vec{k},$$
$$T = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix},$$
$$\vec{k} = \begin{bmatrix} k_0 & k_1 & k_2 \end{bmatrix}^T, \quad (2)$$

where $\vec{x}$ is a physical coordinate vector and $\vec{\xi} = [\xi, \eta, \zeta]^T$ is the canonical coordinate vector of $\vec{x}$. As depicted in Fig. 1, after the coordinate transformation, the first vertex of the cell is located at the origin, and the other three vertices are located one unit away along the three axes, ξ, η, and ζ, of the canonical coordinate system.

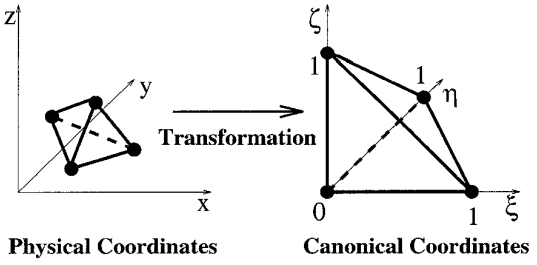

Fig. 1. Coordinate system transformation.

On the other hand, the physical coordinates can be represented as linear functions of the canonical coordinates, too. Without loss of generality, we assume that the vertices of the original cell are ordered as those of the unit cell in Fig. 2. The inverse coordinate transformation function can be expressed as:

$$\vec{x} = (1 - \xi - \eta - \zeta) * \vec{x}_0 + \xi * \vec{x}_1 + \eta * \vec{x}_2 + \zeta * \vec{x}_3, \quad (3)$$

where ξ, η, and ζ are the canonical coordinates and $\vec{x}_i$, $i = 0$, ..., 3 are the physical coordinates of the four vertices.

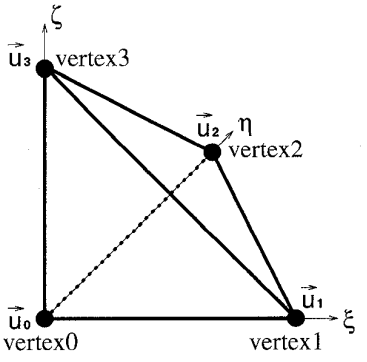

Fig. 2. Unit tetrahedral cell in canonical coordinate system.

3 Vector Field Interpolation

When calculating streamlines in the canonical coordinate system, the velocity field might be transformed from the physical coordinate system to the canonical coordinate system before the calculation is taken place [11]. However, in our work, the velocity field is not transformed. Instead, a

new governing equation of streamlines is deduced in the next section to accommodate the change of coordinate system. Therefore, the three components of the original vector field are treated as three linear functions of the canonical coordinates directly. Referring to the unit cell shown in Fig. 2, the interpolation function of the velocity field can be formulated as:

$$\vec{u}^*\left(\vec{\xi}\right) = (1-\xi-\eta-\zeta) * \vec{u}_0 + \xi * \vec{u}_1 + \eta * \vec{u}_2 + \zeta * \vec{u}_3$$
$$\vec{u}_i = \begin{bmatrix} u_i & v_i & w_i \end{bmatrix}^T,\ i = 0,\ldots,3,$$

where ξ, η, and ζ are the canonical coordinates, and $\vec{u}_i$, $i = 0, \ldots, 3$ are the velocity field values at the vertices. The interpolation function can be expanded as:

$$\vec{u}^*\left(\vec{\xi}\right) = B * \vec{\xi} + \vec{d},$$
$$B = \begin{pmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{pmatrix},$$
$$= \begin{pmatrix} u_1 - u_0 & u_2 - u_0 & u_3 - u_0 \\ v_1 - v_0 & v_2 - v_0 & v_3 - v_0 \\ w_1 - w_0 & w_2 - w_0 & w_3 - w_0 \end{pmatrix},$$
$$\vec{d} = \begin{bmatrix} u_0 & v_0 & w_0 \end{bmatrix}^T. \quad (4)$$

The elements of the matrix B can be computed directly since the vector field values are stored at all vertices.

4 Streamline Construction

Since the velocity field is not transformed from the physical coordinate system to the canonical coordinate system, the original governing equation of streamlines defined in the physical coordinate system needs to be modified. By differentiating both sides of (2) and applying (1), we have:

$$\frac{d\vec{\xi}}{dt} = T * \frac{d\vec{x}}{dt}$$
$$= T * \vec{u}(\vec{x}).$$

From (2), the above equation can be rewritten as:

$$\frac{d\vec{\xi}}{dt} = T * \vec{u}\left(T^{-1} * \left(\vec{\xi} - \vec{k}\right)\right)$$
$$= T * \vec{u}^*\left(\vec{\xi}\right),$$

where $\vec{u}^*$ is the vector field in the canonical coordinate system, which is defined in (4). Therefore the governing equation of streamlines can be written as:

$$\frac{d\vec{\xi}}{dt} = T * \vec{u}\left(\vec{\xi}\right)$$
$$= T * \left(B * \vec{\xi} + \vec{d}\right)$$
$$= C * \vec{\xi} + \vec{e}. \quad (5)$$

To solve (5), the fourth-order Runge-Kutta method is usually applied. After obtaining a new point on the streamline, the cell containing this point needs to be located for further calculation. A cell-searching method is described in the next section. In addition, the physical coordinates of the new point are computed by using (3). The physical coordinates are used for constructing a graphical representation of the streamline. After identifying the cell, the calculation of the streamline can be continued. This procedure is repeated until the streamline reaches a physical boundary or the number of time steps exceeds a pre-defined limit.

4.1 Cell Searching

It is not a straight forward task to identify the cell containing a given streamline point. We use an algorithm presented in [12] to solve this cell searching problem. After a new streamline point is calculated in the canonical coordinate system, the cell containing this point can be located based on the following rules:

- If all the canonical coordinates belong to the interval [0, 1], and their sum ($\xi + \eta + \zeta$) is less than or equal to 1, the point is resident in the current cell and the cell searching is completed.
- If any one of the canonical coordinates is less than 0, the searching continues in the neighboring cell which is in the half space where the canonical coordinate is negative.
- Otherwise, the neighboring cell with coordinate sum greater than 1 is selected to replace the current cell and the searching continues.

A two-dimensional example is shown in Fig. 3.

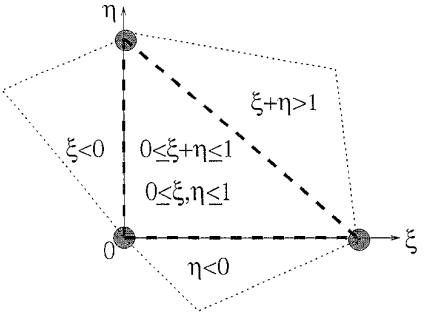

Fig. 3. Cell searching in 2D canonical coordinate system.

4.2 Step Size

In order to integrate a streamline, we need to select a proper discretization step size. In [13], Buning proposed a method to calculate the step size for streamline construction. In his method, the step size is determined by the cell size and the inverse of the velocity vector magnitude. We adopt a similar method to compute the step size in the canonical coordinate system. In our algorithm, we make the distance between two consecutive streamline points in the canonical coordinate system be less than 1, which is the shortest edge of the unit cell in Fig. 2. Assume that the step size is h, the flow velocity values at the four vertices are $\vec{u}_0, \vec{u}_1, \vec{u}_2$, and $\vec{u}_3$. Based on (5), the step size is determined by:

$$\left\|\Delta\vec{\xi}\right\|_2 \simeq h * \left\|T * \vec{u}\left(\vec{\xi}\right)\right\|_2$$
$$\leq h * \max_{0\leq i\leq 3}\left\|T * \vec{u}_i\right\|_2$$
$$\leq 1,$$

where $\|\cdot\|_2$ is the *2-norm* of the vector field. In order to satisfy this inequality, we select:

$$h = \min_{0 \le i \le 3} \frac{1}{\|T * \vec{u}_i\|_2}.$$

The step size h can be computed and stored for each cell based on the local velocity field. In our implementation, a global step size, which is equal to the minimum value of h over all cells, is used for the streamline construction.

5 Streamribbon and Iconic Streamtube Construction

A streamribbon has two edges. The first edge of a streamribbon is the calculated streamline, and the second edge is generated by connecting the end points of the normal vectors of the streamline. The normal vectors are calculated by rotating a constant length vector about the streamline at each point of the streamline. The constant length vector can be any vector which is orthogonal to the streamline at the initial point in the physical coordinate system. Since the streamline construction is performed in the canonical coordinate system, the constant normal vector is transformed into the canonical coordinate system and rotated there. After being rotated, the normal vector is transformed back to the physical coordinate system.

The surface of the streamribbon is formed by connecting the end points of the normal vectors and their corresponding points on the streamline. An example is depicted in Fig. 4. The angle of rotating the constant length vector is governed by:

$$\begin{aligned} \frac{d\theta}{dt} &= \frac{1}{2}(\vec{w} \cdot \vec{s}), \\ \vec{w} &= curl(\vec{u}\,*), \\ \vec{s} &= \frac{\vec{u}\,*}{\|\vec{u}\,*\|}, \end{aligned} \tag{6}$$

where θ is the rotation angle. Equations (6) and (5) are solved stepwise when constructing a streamribbon.

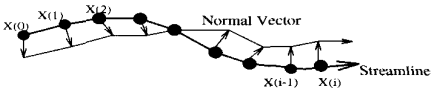

Fig. 4. Example of streamribbon construction.

An iconic streamtube is created by generating a streamline and by connecting the circular crossflow sections along the streamline. The radius of an iconic streamtube, r, is governed by the following ordinary differential equation:

$$\frac{1}{r}\frac{dr}{dt} = \frac{1}{2}\nabla_T \cdot \vec{u}\,*, \tag{7}$$

where $\nabla_T \cdot \vec{u}\,*$ is the local cross flow divergence and is defined as:

$$\nabla_T \cdot \vec{u}* = \nabla \cdot \vec{u}\,* - \frac{du\,*'}{d\xi'},$$

in which $\frac{du*'}{d\xi'}$ represents the change of velocity magnitude along the streamline. This ordinary differential equation says that the change rate of the local flow volume is equivalent to the local flow divergence. Equations (5) and (7) are solved stepwise when constructing an iconic streamtube. Equation (5) is used to calculate the center of the tube, while (7) is used to calculate the tube radius. Fig. 5 contains an example of constructing an iconic streamtube.

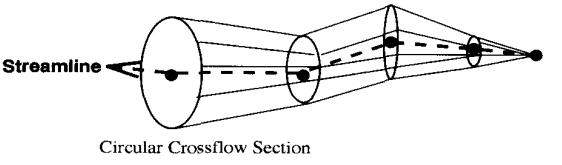

Fig. 5. Example of iconic streamtube construction.

5.1 The Curl and Divergence of the Vector Field

In the canonical coordinate system, the partial derivative $\frac{\partial f}{\partial \xi}$ of a linear function f in a unit cell can be calculated by using the following equation:

$$\begin{aligned} \frac{\partial f}{\partial \xi} &= \frac{f_1 - f_0}{\xi_1 - \xi_0} \\ &= f_1 - f_0, \end{aligned}$$

where f_0 and f_1 are the function values at vertices 0 and 1 of the cell, respectively, and ξ_0 and ξ_1 are the ξ coordinates of these two vertices. By applying the same deduction steps, we have:

$$\begin{aligned} \frac{\partial f}{\partial \eta} &= f_2 - f_0, \\ \frac{\partial f}{\partial \zeta} &= f_3 - f_0, \end{aligned}$$

where f_2 and f_3 are the function values at vertices 2 and 3. Assume $\vec{u}* = [u*, v*, w*]^T$ is the vector field defined in the canonical coordinate system. Equation (4) implies:

$$\left[\frac{\partial \vec{u}\,*}{\partial \vec{\xi}}\right] = B.$$

Therefore the *divergence* and *curl* of $\vec{u}\,*$ can be calculated as:

$$\begin{aligned} \nabla \cdot \vec{u}* &= \frac{\partial u\,*}{\partial \xi} + \frac{\partial v\,*}{\partial \eta} + \frac{\partial w\,*}{\partial \zeta} \\ &= b_{00} + b_{11} + b_{22} \\ &= (u_1 - u_0) + (v_2 - v_0) + (w_3 - w_0), \end{aligned} \tag{8}$$

$$\begin{aligned} curl(\vec{u}\,*) &= \left[\frac{\partial w\,*}{\partial \eta} - \frac{\partial v\,*}{\partial \zeta}, \frac{\partial u\,*}{\partial \zeta} - \frac{\partial w\,*}{\partial \xi}, \frac{\partial v\,*}{\partial \xi} - \frac{\partial u\,*}{\partial \eta}\right]^T \\ &= \big[(w_2 - w_0) - (v_3 - v_0), \\ &\quad (u_3 - u_0) - (w_1 - w_0), \\ &\quad (v_1 - v_0) - (u_2 - u_0)\big]^T. \end{aligned} \tag{9}$$

6 New Computational Methods

In order to create streamlines, streamribbons and iconic streamtubes, we need to solve the ordinary differential equations mentioned in the previous sections. Based on the interpolation functions of a linear tetrahedral cell, we develop explicit solutions of the ordinary differential equations to speed up the computation.

6.1 A Specialized Version of the Runge-Kutta Method

The governing equation of a streamline in canonical coordinate system, as given in (5), is:

$$\frac{\partial \vec{\xi}(t)}{dt} = f(\vec{\xi}, t) = C * \vec{\xi} + \vec{e}$$

The fourth-order Runge-Kutta method is given by:

$$\begin{aligned}
\vec{\xi}(t+h) &= \vec{\xi}(t) + \frac{1}{6}(F_1 + 2F_2 + 2F_3 + F_4),\\
F_1 &= hf(\vec{\xi}, t),\\
F_2 &= hf(\vec{\xi} + F_1/2, t + h/2),\\
F_3 &= hf(\vec{\xi} + F_2/2, t + h/2),\\
F_4 &= hf(\vec{\xi} + F_3, t + h).
\end{aligned}$$

By replacing f with $C * \vec{\xi} + \vec{e}$, $F1$, $F2$, $F3$, and $F4$ can be expanded as:

$$\begin{aligned}
F_1 &= hf(\vec{\xi}, t)\\
&= h(C\vec{\xi} + \vec{e}),\\
F_2 &= hf(\vec{\xi} + F_1/2, t + h/2)\\
&= h(C(\vec{\xi} + F_1/2) + \vec{e})\\
&= (h^2C/2 + h)(C\vec{\xi} + \vec{e}),\\
F_3 &= hf(\vec{x} + F_2/2, t + h/2)\\
&= h(C(\vec{\xi} + F_2/2) + \vec{e})\\
&= (h^3C^2/4 + h^2C/2 + h)(C\vec{\xi} + \vec{e}),\\
F_4 &= hf(\vec{\xi} + F_3, t + h)\\
&= h(C(\vec{\xi} + F_3) + \vec{e})\\
&= (h^4C^3/4 + h^3C^2/2 + h^2C + h)(C\vec{\xi} + \vec{e}).
\end{aligned}$$

By using these equations, the Runge-Kutta method can be expressed as:

$$\begin{aligned}
\vec{\xi}(t+h) &= \vec{\xi}(t) + \frac{1}{6}(F_1 + 2F_2 + 2F_3 + F_4)\\
&= \left(I + hC + \frac{(hC)^2}{2!} + \frac{(hC)^3}{3!} + \frac{(hC)^4}{4!}\right)\vec{\xi}(t) +\\
&\quad h\left(I + \frac{hC}{2!} + \frac{(hC)^2}{3!} + \frac{(hC)^3}{4!}\right)\vec{e}.
\end{aligned}$$

Therefore,

$$\vec{\xi}(t+h) = H_1\vec{\xi}(t) + \vec{e}_1, \tag{10}$$

where

$$\begin{aligned}
H_1 &= I + hC + \frac{(hC)^2}{2!} + \frac{(hC)^3}{3!} + \frac{(hC)^4}{4!},\\
\vec{e}_1 &= h\left(I + \frac{hC}{2!} + \frac{(hC)^2}{3!} + \frac{(hC)^3}{4!}\right)\vec{e}.
\end{aligned}$$

Since C and $\vec{e}$ are constants, H_1 and $\vec{e}_1$ can be calculated by using Horner's algorithm [14] and stored in the preprocessing stage. Hereafter the computations of the fourth-order Runge-Kutta method require only a matrix-vector multiplication and a vector-vector addition.

6.2 Explicit Solution for Computing the Angular Rotation Rate

The angular rotation rate is governed by the ODE formulated in (6). Since the velocity $\vec{u}^*$ is linear within a cell, the curl of $\vec{u}^*$ is a constant vector, which can be calculated by using (9):

$$\begin{aligned}
\vec{w} &= curl(\vec{u})\\
&= [(w_2 - w_0) - (v_3 - v_0),\\
&\quad (u_3 - u_0) - (w_1 - w_0),\\
&\quad (v_1 - v_0) - (u_2 - u_0)]^T.
\end{aligned}$$

By using the Trapezoid rule, (6) can be solved explicitly:

$$\begin{aligned}
\frac{d\theta}{dt} &= \frac{1}{2}(\vec{w} \cdot \vec{s}),\\
\int_0^h \frac{d\theta}{dt}dt &= \frac{1}{2}\vec{w} \cdot \int_0^h \vec{s}dt,\\
\theta(h) - \theta(0) &= \frac{1}{2}\vec{w} \cdot (\vec{s}(\vec{\xi}_h) + \vec{s}(\vec{\xi}_0))\frac{h}{2},\\
\theta(h) &= \theta(0) + \frac{h}{4}\vec{w} \cdot \left(\frac{\vec{u}^*(\vec{\xi}_h)}{\|\vec{u}^*(\vec{\xi}_h)\|} + \frac{\vec{u}^*(\vec{\xi}_0)}{\|\vec{u}^*(\vec{\xi}_0)\|}\right)
\end{aligned}$$

where $\theta(0)$ is the rotation angle at the previous time step, $\theta(h)$ is the rotation angle at the current time step, $\vec{u}^*(\vec{\xi}_h)$ is the velocity at the current time step, $\vec{u}^*(\vec{\xi}_0)$ is the velocity at the previous time step, and h is the time step size. This closed form solution is used to compute the rotation angle of the normal vector about the streamline. The only unknown values involved in this solution are $\vec{u}^*(\vec{\xi}_h)$ and its velocity magnitude. Once a new point $\vec{\xi}_h$ of a streamline is computed, $\vec{u}^*(\vec{\xi}_h)$ can be calculated by using (4). The major computational costs of this solution include a vector field interpolation, a vector-vector addition and a vector inner product.

6.3 Explicit Solution for Computing the Radius

The governing equation of the iconic streamtube radius is shown in (7). This ODE can be solved analytically:

$$\int_{r_0}^{r_h}\frac{1}{r}dr = \frac{1}{2}\int_0^h \nabla_T \cdot \vec{u}^{\,*}\,dt,$$

$$\ln(r_h) - \ln(r_0) = \frac{1}{2}\int_0^h \nabla_T \cdot \vec{u}^{\,*}\,dt,$$

$$\ln(r_h) = \ln(r_0) + \frac{1}{2}\left(\int_0^h \nabla \cdot \vec{u}^{\,*}\,dt - \int_0^h \frac{du^{*'}}{d\xi'}dt\right).$$

From (8), the divergence of $\vec{u}^{\,*}$ is a constant:

$$\nabla \cdot \vec{u}^{*} = b_{00} + b_{11} + b_{22},$$

and

$$d\xi' = u^{*'}\,dt.$$

Therefore,

$$\ln(r_h) = \ln(r_0) + \frac{1}{2}\left((b_{00} + b_{11} + b_{22})h - \int_0^h \frac{du^{*'}}{u^{*'}}\right),$$

$$r_h = r_0 \exp\left(\frac{1}{2}\left((b_{00} + b_{11} + b_{22})h - \ln\left(u_h^{*'}\right) + \ln\left(u_0^{*'}\right)\right)\right)$$

$$r_h = r_0 \exp\left(\frac{1}{2}(b_{00} + b_{11} + b_{22})h\right)\sqrt{\frac{u_0^{*'}}{u_h^{*'}}},$$

where r_h is the iconic streamtube radius at the current step, r_0 is the radius at the previous step, and $u_0^{*'}$ and $u_h^{*'}$ are the velocity magnitudes at the previous step and the current step. Since there is no unknown value in the right hand side of the equation, the cost of calculating r_h is composed of only a few scalar multiplications and two function calls. From this equation, we can see that faster velocity results in a smaller radius. For incompressible flow, where the divergence of the velocity is zero, our result is equivalent to that obtained in [7].

7 Implementation

The major data structures of our program contain a list of cell records and a list of vertex records. To further speed up the computation, we precompute and store the connectivities among data cells along with the coefficients of the coordinate transformation function and the coefficients of the specialized Runge-Kutta method. Consequently, a cell record contains the coefficients of the coordinate transformation function, the coefficients of the specialized Runge-Kutta method, the four vertex indices of this cell, and the four indices of cells which are adjacent to this cell. Thus 24 floats and eight integers are stored in a cell record, and the size of a cell record is 128 bytes. The values stored in a vertex record include the physical coordinates of the vertex and the velocity value at the vertex.

In order to study the performance of the specialized Runge-Kutta, SRK4, method, we have also implemented the second-order, RK2, and the fourth-order, RK4, methods. The cell connectivity information and the coordinate transformation function used in the SRK4 method are also used by the RK2 and the RK4 methods to make the comparison fair. Thus 12 floats and eight integers are stored in a cell record for the RK2 and the RK4 methods. The size of a cell record is 80 bytes.

8 Test Results

Three data sets are used in our testing. The first data set is artificially created. It contains 68,921 vertices uniformly positioned in a cube and 320,000 tetrahedra. The memory requirement for this data set is about 40 Mega-bytes. The vector field at a vertex is calculated by evaluating three linear functions:

$$u(x, y, z) = -0.5x - 6.0y,$$
$$v(x, y, z) = 6.0x - 0.5y,$$
$$w(x, y, z) = -2.0z + 20.5.$$

The second data set is the blunt fin data set provided by researchers at the NASA Ames Research Center. It is generated from a computational fluid dynamics simulation of air flow over a flat plate with a blunt fin rising from the plate. The flow is symmetrical about a plane through the center of the fin, so only one half of the complete geometry is present. Note that the data set is originally on a structured curvilinear grid. We convert it into an unstructured grid by splitting each hexahedron into six tetrahedra. The resulting unstructured-grid data set contains 224,874 tetrahedral cells and 40,960 vertices. This data set requires 28 Mega-bytes memory space. The third data set is provided by Dr. Mavriplis at ICASE. It is obtained from a computational fluid dynamics simulation of transonic flow about an ONERA-M6 wing. The free-stream Mach number equals 0.84, and the angle of attack is 3.06 degrees. There are 287,962 tetrahedral cells and 53,961 vertices in this data set. The memory requirement for this data set is about 36 Mega-bytes.

Each test begins by randomly selecting one hundred initial points. Then the corresponding streamlines are constructed. The maximum number of time steps for each streamline construction is set to 5,000. An IBM RS-6000 Model 560 workstation is used for our testing.

8.1 Canonical Coordinate System Versus Physical Coordinate System

In our previous research [9], similar formulations were used but most of the calculations, except cell searching, were carried out in the physical coordinate system. It is interesting to compare computational cost between the new and the old implementations of the SRK4 method. In the old implementation, the size of a cell record is 176 bytes. In the new implementation the size of a cell record is reduced to 128 bytes. To derive accurate numbers for comparing the performance of the SRK4 method in the two different coordinate systems, we have rerun the tests for the previous implementation by using a more careful timing procedure. The average cost of computing one streamline point is measured for the two SRK4 programs. This average cost includes the calculation of the coordinates of a streamline point, the cell searching operation and the coordinate transformation. The timing results, in μs, are given in Table 1.

TABLE 1
EXECUTION TIME OF A SINGLE STEP OF SRK4

time (μs)	Canonical Coord.	Physical Coord.
data set 1	11.53	22.16
data set 2	10.46	17.49
data set 3	13.28	17.43

The new SRK4 program is always faster than the old SRK4 program. We find that the cell searching operation makes the old program slower. About 60% more searching is done by the old SRK4 implementation. In the new implementation, streamline points are calculated in the canonical coordinate system. Based on the canonical coordinates of a streamline point, it is easier to verify whether a new cell should be searched. Therefore unnecessary cell searching calculation is avoided.

8.2 SRK4 Method Versus RK4 and RK2 Methods

The execution times, in seconds, of constructing from 1 to 100 streamlines by using the SRK4, the RK4, and the RK2 methods are plotted in Figs. 6, 7, and 8. The execution time includes the cost of streamline point calculation, cell searching and coordinate transformation. To make the comparison fair, the RK2 and the RK4 programs are implemented based on the framework of the SRK4 program. That is, all information used by the SRK4 program is also used in the RK2 and the RK4 programs. Some effort is undertaken to optimize the RK2 and RK4 implementation; for example, if the computations at the current step and the previous step occur in the same cell, all interpolation function coefficients calculated for the previous step are reused in the computation of the current step.

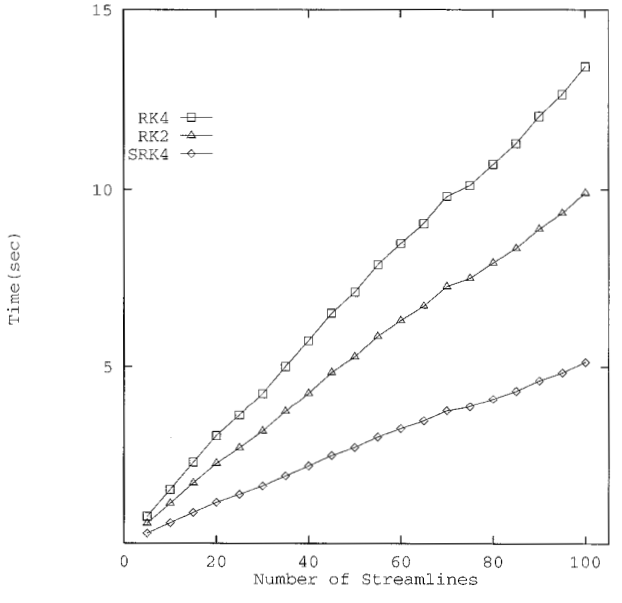

Fig. 6. Timing of constructing streamlines on data set 1.

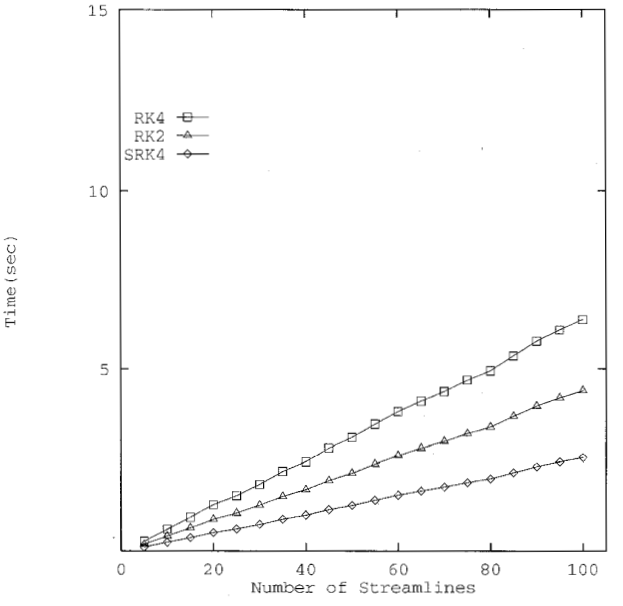

Fig. 7. Timing of constructing streamlines on data set 2.

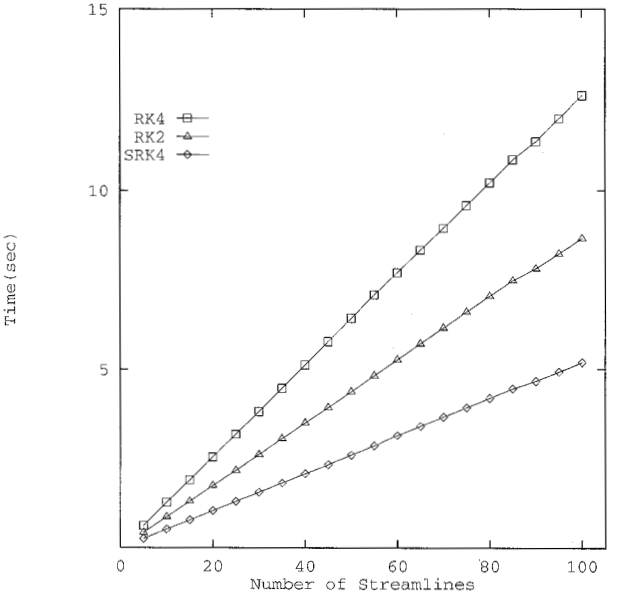

Fig. 8. Timing of constructing streamlines on data set 3.

Since the major function evaluations of all three methods are of the same kind, i.e., matrix-vector multiplication, we can predict their performance by counting the number of function evaluations used in each method. For calculating a new point in a streamline, only one function evaluation is needed for the SRK4 method, while four function evaluations are needed for the RK4 method and two function evaluations are needed for the RK2 method. Therefore, the SRK4 method is supposed to be faster than the RK2 method by a factor of 2.0, and faster than the RK4 method by a factor of 4.0. Since the costs of cell searching and coordinate transformation are included in the cost, and the RK2 and

the RK4 programs are optimized to avoid unnecessary calculation, the performance of the SRK4 method depicted in the figures is not that much better. However, it is still at least 1.7 times faster than the RK2 method and 2.5 times faster than the RK4 method. The average cost of computing a streamline point is presented in Table 2.

TABLE 2
EXECUTION TIME OF A SINGLE TIME STEP OF SRK4 AND RK2 AND RK4 METHODS

time (μs)	SRK4	RK2	RK4
data set 1	11.53	22.24	30.06
data set 2	10.46	17.86	25.94
data set 3	10.48	17.46	25.51

We also implemented the RK2 and the RK4 methods without optimization, i.e. all interpolation function coefficients are re-calculated at each step. Test results of these two methods are compared with those of the SRK4 method and shown in Fig. 9 for the first data set. According to the results, the SRK4 program is now much faster than the RK2 and the RK4 programs. Specifically, the SRK4 method is at least 4 times faster than RK2 method and 5 times faster than RK4 method. Table 3 shows the average cost of a single step computation. The purpose of this test is to reveal the superiority of the SRK4 method over the RK2 and the RK4 methods in an extreme situation, in which no two consecutive points of a streamline locate at the same cell.

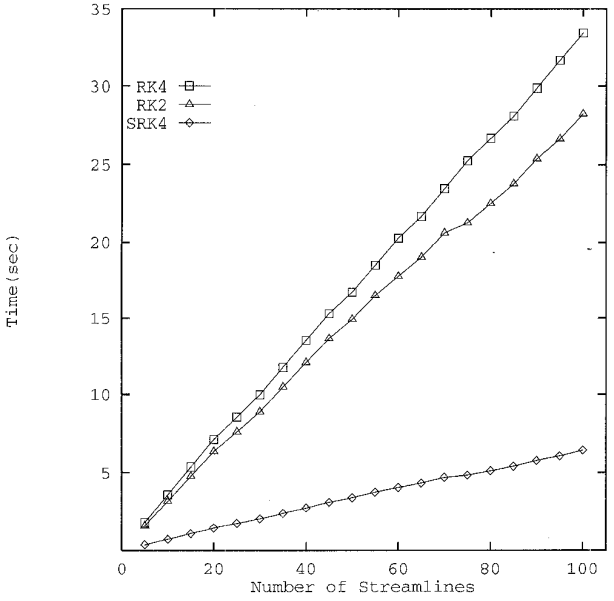

Fig. 9. Testing results by using data set 1 with nonoptimized codes.

TABLE 3
EXECUTION TIME OF A SINGLE STEP OF THE SRK4, (NONOPTIMIZED) RK2 AND (NONOPTIMIZED) RK4 METHODS

time (μs)	SRK4	RK2	RK4
data set 1	11.53	49.44	55.44
data set 2	10.46	48.49	54.73
data set 3	10.48	49.41	55.69

8.3 Visualization Results

Some visualization results generated by our programs are displayed in the two color plates. The upper left image in Plate 1 shows a streamribbon visualization of the first data set. In this image, the streamribbons spiral toward a critical point which is a saddle point in the vector field. The streamribbons are colored according to velocity magnitudes. The upper right image shows an iconic streamtube visualization of the same data set. This image reveals not only the rotation of the flow but also the expansion and contraction of the flow. The streamribbon and iconic streamtube visualization of the blunt fin data set are shown in the lower half of Plate 1. Some irregular flow movements have been revealed near the vertical boundary of the physical domain.

The pair of the images in the upper half of Plate 2 display the streamribbon and iconic streamtube visualization of the ONERA-M6 wing data set. We can see that the flow rotates around the wing tip while it moves smoothly before and after passing of the wing tip. Finally, the bottom two images in Plate 2 show the streamribbon and the iconic streamtube images of a fourth data set provided by researchers at the NASA Langley Research Center. This data set is generated from a computational fluid dynamics simulation of flow about a wing with a trailing-edge flap in a wind tunnel. There are about two million cells in this data set. Fifty streamlines are constructed by using fifty initial points which are randomly selected in the physical domain. From this image, we can see that the flow pattern is changed when the flow passes above the deflected trailing-edge flap. A secondary flow that takes place through the gap between the wing and the flap is shown in this image too. For a large data set containing several millions of cells, a large number of streamlines can be traced and stored in a batch mode or by using a parallel computer. Researchers then examine the stored streamlines using an interactive viewer.

9 CONCLUSIONS

The plotting of streamlines is an effective way of visualizing fluid motion in steady flow. We derive a new computational method which greatly simplifies the conventional fourth-order Runge-Kutta method. Explicit solutions are derived for computing the angular rotation rate of flow and the flow expansion rate to speed up the construction of streamribbons and iconic streamtubes, respectively. In order to simplify mathematical formulations, reduce memory usage and further improve the performance of our specialized RK4 method, the computation is performed in the canonical coordinate system instead of the physical coordinate system. These new computational methods and their implementation are evaluated by using three data sets. Test results show that the SRK4 method is indeed superior to the more conventional methods. Moreover, the explicit solutions derived also significantly reduce the overall execution time.

While we have improved particle tracing calculations, the computational and memory requirements of tracing in a large data set are high. For example, when using our algo-

rithm, a data set of three million cells would take over 350 megabytes of memory. For such a data set which do not fit into the main memory of an average workstation, we have developed an out-of-core tracing strategy with which all data are stored on disk and only a subset of the data is loaded into the main memory upon demand. A prototype implementation of the out-of-core approach demonstrates interactive particle tracing in very large data sets on a 64-megabyte workstation. On the other hand, if a parallel computer is available, both data and tracing can be distributed to multiple processors.

Acknowledgments

This work has been supported in part by the Advanced Combustion Engineering Research Center and the National Aeronautics and Space Administration under NASA contract NAS1-19480. Thanks go to the anonymous reviewers who provided many useful suggestions on ways to improve the manuscript.

References

[1] J.K. Vennard and R.L. Street, *Elementary Fluid Mechanics*. John Wiley & Sons, 1975.
[2] G. Volpe, "Streamlines and Streamribbons in Aerodynamics," *Proc. AIAA 27th Aerospace Science Meeting*, Reno, Nev., Jan. 1989.
[3] J. Hultquist, "Interactive Numerical Flow Visualization Using Stream Surfaces," PhD thesis, Univ. of North Carolina, Chapel Hill, Technical Report TR95-014, 1995.
[4] D. Darmofal and R. Haimes, "Visualization of 3-D Vector Fields: Variations on a Stream," *Proc. AIAA 30th Aerospace Science Meeting and Exhibit*, Reno, Nev., Jan. 1992.
[5] K.-L. Ma and P.J. Smith, "Cloud Tracing in Convection-Diffusion Systems," *Proc. Visualization '93 Conf.*, pp. 253-260. IEEE CS Press, 1993.
[6] H.-G. Pagendarm and B. Walter, "Feature Detection from Vector Quantities in a Numerically Simulated Hypersonic Flow Field in Combination with Experimental Flow Visualization," *Proc. Visualization '94 Conf.*, pp. 117-123. IEEE CS Press, 1994.
[7] W.J. Schroeder, C.R. Volpe, and W.E. Lorensen, "The Stream Polygon: A Technique for 3D Vector Field Visualization," *Proc. Visualization '91 Conf.*, pp. 126-132. IEEE CS Press, 1991.
[8] M.P. Garrity, "Raytracing Irregular Volume Data," *Proc. San Diego Workshop Volume Visualization*, pp. 35-40, San Diego, Dec. 1990.
[9] S.K. Ueng, K. Sikorski, and K.-L. Ma, "Fast Algorithms for Visualizing Fluid Motion in Steady Flow on Unstructured Grids," *Proc. Visualization '95 Conf.*, pp. 313-320. IEEE CS Press, 1995.
[10] A. Sadarjoen, T. van Walsum, A.J.S. Hin, and F.H. Post, "Particle Tracing Algorithms for Curvilinear Grids," *Proc. Fifth Eurographics Workshop Visualization in Scientific Computing*, Rostock, Germany, May 1994.
[11] B. Hamann, D. Wu, and R. Moorhead, "Flow Visualization with Surface Particles," *IEEE Trans. Visualization and Computer Graphics*, vol. 1, no. 3, pp. 210-217, Sept. 1995.
[12] R. Lohner and J. Ambrosiano, "A Vectorized Particle Tracer for Unstructured Grids," *J. Computational Physics*, vol. 91, pp. 22-31, 1990.
[13] P. Buning, "Sources of Error in the Graphical Analysis of cfd Results," *J. Scientific Computing*, vol. 3, no. 2, pp. 149-164, 1988.
[14] G.H. Golub and C.F. van Loan, *Matrix Computations*. Johns Hopkins Univ. Press, 1989.

Shyh-Kuang Ueng received his BS from Soochow University, Taipei, Taiwan, in 1985, and his MS in computer science from National Taiwan University, Taipei, Taiwan, in 1988. He is currently a PhD student in computer science at the University of Utah. His research interests include scientific computing, scientific visualization, and algorithms.

Christopher Sikorski holds the PhD degree from the University of Utah in scientific computation with a special emphasis on computational complexity and algorithm design. He is an associate professor of computer science at the University of Utah. His research interests concentrate on parallel scientific computation, computational complexity for continuous problems, 3D visualization algorithms, and applications in computational fluid dynamics, combustion engineering, and geophysics modeling.

Dr. Sikorski is the author or coauthor of two research monographs, one lecture notes, numerous journal articles, and several papers in conference proceedings. His graduate students are or were holding positions at Columbia University, the University of Utah, Lehigh University, National Taiwan University, National Korean University, and NASA Langley Research Center.

Kwan-Liu Ma received his BS, MS, and PhD degrees in computer science from the University of Utah. He is a staff scientist at the Institute of Computer Applications for Science and Engineering (ICASE) and an adjunct assistant professor of computer science at Old Dominion University, where he teaches scientific visualization. His research interests include computer graphics, scientific visualization, and parallel and distributed computing. Dr. Ma is a member of the IEEE, ACM, and Phi Kappa Phi.

PLATE 1