Fast Algorithms for Visualizing Fluid Motion in Steady Flow on Unstructured Grids

S.K. Ueng and K. Sikorski
Department of Computer Science
University of Utah
Salt Lake City, Utah 84112

Kwan-Liu Ma
ICASE, Mail Stop 132C
NASA Langley Research Center
Hampton, Virginia 23681

Abstract

The plotting of streamlines is an effective way of visualizing fluid motion in steady flows. Additional information about the flowfield, such as local rotation and expansion, can be shown by drawing in the form of a ribbon or tube. In this paper, we present efficient algorithms for the construction of streamlines, streamribbons and streamtubes on unstructured grids. A specialized version of the Runge-Kutta method has been developed to speed up the integration of particle pathes. We have also derived close-form solutions for calculating angular rotation rate and radius to construct streamribbons and streamtubes, respectively. According to our analysis and test results, these formulations are two to four times better in performance than previous numerical methods. As a large number of traces are calculated, the improved performance could be significant.

1 Introduction

Streamlines, streamribbons and streamtubes are very powerful techniques for visualizing steady vector fields. A streamline is the path of a massless particle which is released in a steady flow. The plotting of the particle paths produces a streamline picture, which is of both qualitative and quantitative value to the engineer. Streamline pictures allow the engineer to visualize fluid motion and to locate regions of high and low velocity and, from these, zones of high and low pressure.

Given a fluid flow with velocity field $\vec{u}(\vec{x}(t))$, a streamline is an integral curve of $\vec{u}$. That is, a streamline can be calculated by solving the following equation:

$$\frac{d\vec{x}(t)}{dt} = \vec{u}(\vec{x}(t)) \qquad (1)$$

where t is a parameter along the streamline and is not to be confused with time [11].

A streamribbon can show the translation, angular rotation, and rates of shear deformation of the flow. Ideally, it is constructed by tracing a set of streamlines originated from multiple seed locations on a straight line segment. That is, the path swept by the deformable line segment becomes a streamribbon. Volpe [12] constructs a streamribbon in this fashion 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. In practice, the construction of streamribbons is simplified, though some information such as shear deformation would be lost. In [4], a streamribbon is generated by computing only a few streamlines and creating polygons between adjacent streamlines to form the surface of the streamribbon. This method still requires complicated algorithms to deal with the convergence, the divergence and the splitting of streamribbons. Darmofal and Haimes [2], Ma and Smith [7], and Pargendarm [9] use one streamline and vectors normal to the local velocity to form a streamribbon. In this way, the resulting ribbons only show the translation and angular rotation of the flow. We adopt Darmofal and Haimes' algorithm by using two parallel edges to form a streamribbon. First, a streamline is generated to serve as the first edge of the streamribbon. A normal vector is calculated at each point of the streamline by rotating a constant length vector about the streamline. Then the second edge of the streamribbon is formed by connecting the end points of the normal vectors.

Formally, a streamtube is defined as the surface formed by all streamlines passing through a given closed curve in the flow [11]. Streamtubes are used to visualize expansion, contraction and deformation of the flow. In [2], 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. A streamtube constructed in this manner does not reveal the deformation of the flow. Again, this is a technique more computational

(See color plates, page CP-37)

1070-2385/95 $4.00 © 1995 IEEE

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE

feasible and we adopt it in this work. In [7], 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 root mean square value for the vector field and its Larangian time scale. The result defines the radius of the cross flow section and also form a tube-like surface. Schroeder et. al. [10] introduce a technique called Stream Polygon for visualizing local deformation of the flow.

In this paper, we present efficient algorithms to compute streamlines, streamribbons and streamtubes on unstructured grids. Our algorithms are mainly based on those developed in [2]. Several new computational techniques are derived and used to improve performance. These new computational techniques include a specialized version of Runge-Kutta method, a simpler procedure to compute the angular rotation rate of the flow and an explicit solution for calculating the radius of streamtube. An overview of our algorithms is described in Section 2. The new computational techniques are derived in Section 3. The data structures used and the memory requirements for implementing the algorithms for testing are described in Section 4. Finally, we present some experimental results using three different data sets to demonstrate the time efficiency of the new particle tracing algorithm.

2 Overview of the Algorithms

In this paper, we assume that all cells are tetrahedra. Other types of cells have to be decomposed into tetrahedra in preprocessing stages. In a tetrahedral cell, the three components of the vector field are linear functions of the physical coordinates. Their interpolation functions can be formulated as:

$$\begin{aligned} u_1(x,y,z) &= a_1x + b_1y + c_1z + d_1, \\ u_2(x,y,z) &= a_2x + b_2y + c_2z + d_2, \\ u_3(x,y,z) &= a_3x + b_3y + c_3z + d_3. \end{aligned} \tag{2}$$

where $u_i, i = 1,2,3$, are the three components of vector field; $a_i, b_i, c_i, d_i, i = 1,2,3$, are the coefficients of the interpolation functions; x, y, z are the physical coordinates. The above equations can be re-written in a concise form:

$$\vec{u}(\vec{x}) = B\vec{x} + \vec{d} \tag{3}$$

$$B = \begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{pmatrix} \tag{4}$$

$$\vec{d} = [\ d_1 \quad d_2 \quad d_3\ ]^T \tag{5}$$

When calculating a streamline, it is necessary to find the cell in which this streamline enters at each time step. A method is given in [6] to solve this problem. In this method, the physical coordinates of the point calculated at each time step are transformed into the canonical coordinates as shown in Figure 1. Then the canonical coordinates are used to determine the cell which the streamline enters. In this work, we adopt a simpler method to convert the physical coordinates into the canonical coordinates:

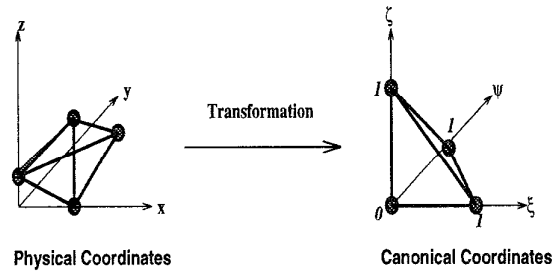

Figure 1: Coordinate System Transformation

$$\vec{\xi} = R\vec{x} + \vec{k} \tag{6}$$

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \tag{7}$$

$$\vec{k} = [\ k_1 \quad k_2 \quad k_3\ ]^T \tag{8}$$

where $\vec{x}$ is a physical coordinate vector and $\vec{\xi}$ is the canonical coordinate vector of $\vec{x}$.

2.1 Streamline Construction

Given an initial point in a physical domain, a streamline can be calculated by solving Equation 1. The 4th order Runge-Kutta method is applied to integrate the equation stepwise. After calculating a point of the streamline, Equation 6 is used to transform the physical coordinates of the point into the canonical coordinates. If all the three components of the canonical coordinates are between 0.0 and 1.0, this point is still inside the current cell where the computation of the point takes place. The coefficients of the interpolation functions of the current cell are still valid for next step integration. Otherwise, a searching for a new cell which contains the point is started according to the canonical coordinates. After finding the new cell, the computation of next position can be performed. This pattern of calculation is repeated until the streamline reaches a physical boundary or the number of time steps exceeds a pre-defined limit.

2.2 Streamribbon Construction

A streamribbon has two edges as we have described. The first edge of a streamribbon is constructed by calculating a 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

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE

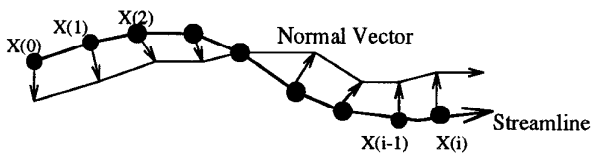

Figure 2: Example of Streamribbon Construction

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. The surface of the streamribbon is then formed by connecting the end points of the normal vectors and their corresponding points on the streamline. An example is depicted in Figure 2. The angle of rotating the constant length vector is governed by:

$$\frac{d\theta}{dt} = \frac{1}{2}(\vec{w} \cdot \vec{s}) \quad (9)$$

$$\vec{w} = curl(\vec{u}) \quad (10)$$

$$\vec{s} = \frac{\vec{u}}{||\vec{u}||} \quad (11)$$

where θ is the rotation angle. Equations 9 and 1 are solved stepwise when constructing a streamribbon.

2.3 Streamtube Construction

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

$$\frac{1}{r}\frac{dr}{dt} = \frac{1}{2}\nabla_T \cdot \vec{u} \quad (12)$$

$$\nabla_T \cdot \vec{u} = \nabla \cdot \vec{u} - \frac{du'}{dx'} \quad (13)$$

where r is the streamtube radius, $\nabla_T \cdot \vec{u}$ is the local cross flow divergence, and $\frac{du'}{dx'}$ represents the change of velocity magnitude along the streamline. Equations 1, 9 and 12 are solved stepwise when constructing a streamtube. Equation 1 is used to calculate the center of the streamtube, while Equations 9 and 12 are used to calculate the angle of rotation and the radius of the streamtube. Figure 3 contains an example of constructing a streamtube.

3 New Computational Methods

In order to construct streamlines, streamribbons and streamtubes, we need to solve the ODE's mentioned in the previous sections. Based on the interpolation functions of linear tetrahedral cell, we had developed specialized ODE solvers to speed up our algorithms.

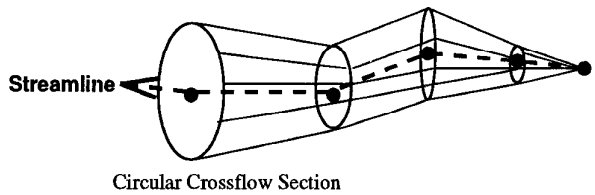

Figure 3: Example of Streamtube Construction

3.1 A Specialized Version of the Runge-Kutta Method for Streamline Construction

By combining Equations 1 and 3 the governing equation of a streamline can be formulated as:

$$\frac{d\vec{x}(t)}{dt} = f(\vec{x}, t) = B\vec{x} + \vec{d} \quad (14)$$

The 4th order Runge-Kutta method is applied to solve this ODE:

$$\vec{x}(t+h) = \vec{x}(t) + \frac{1}{6}(F_1 + 2F_2 + 2F_3 + F_4) \quad (15)$$

$$F_1 = hf(\vec{x}, t) \quad (16)$$

$$F_2 = hf(\vec{x} + F_1/2, t + h/2) \quad (17)$$

$$F_3 = hf(\vec{x} + F_2/2, t + h/2) \quad (18)$$

$$F_4 = hf(\vec{x} + F_3, t + h) \quad (19)$$

where h is the time step size. By substituting Equations 14 and 3 into the right hand sides, Equations 16 - 19 can be expanded as:

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

By using these equations, the Runge-Kutta method shown in Equation 15 can be expressed as:

$$\begin{aligned}
\vec{x}(t+h) &= \vec{x}(t) + \frac{1}{6}(F_1 + 2F_2 + 2F_3 + F_4) \\
&= (I + hB + \frac{(hB)^2}{2!} + \frac{(hB)^3}{3!} + \frac{(hB)^4}{4!})x(\vec{t}) \\
&\quad + h(I + \frac{hB}{2!} + \frac{(hB)^2}{3!} + \frac{(hB)^3}{4!})\vec{d}
\end{aligned}$$

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE

$$= H_1\vec{x}(t) + H_2\vec{d} \quad (20)$$
$$= H_1\vec{x}(t) + \vec{d_1} \quad (21)$$

Since B and $\vec{d}$ are constants, H_1 and $\vec{d_1}$ can be calculated by using Horner's algorithm [3]. Hereafter the computations of the 4th order Runge-Kutta method require only a matrix-vector multiplication and a vector-vector addition.

3.2 Explicit Solution for the Angular Rotation Rate

The angular rotation rate is governed by the ODE formulated in Equation 9. Since the velocity $\vec{u}$ is linear within a cell, the curl of $\vec{u}$ is a constant vector. According to Equation 3, we have:

$$\begin{aligned}\vec{w} &= curl(\vec{u}) \\ &= \nabla \times \vec{u} \\ &= \nabla \times (B\vec{x} + \vec{d}) \\ &= [\ b_3 - c_2 \quad c_1 - a_3 \quad a_2 - b_1\ ]^T\end{aligned}$$

Then Equation 9 can be solved analytically:

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

where $\theta(0)$ is the rotation angle at the previous time step, $\theta(h)$ is the rotation angle at the current time step, $\vec{u}(h)$ is the velocity at the current time step, and $\vec{u}(0)$ is the velocity at the previous time step. 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 $\vec{u}(h)$ and its velocity magnitude. Since $\vec{u}(h)$ can be calculated by using Equation 3, the major cost of this solution is reduced to a matrix-vector multiplication.

3.3 Explicit Solution for the Radius of Streamtube

The governing equation of streamtube radius is shown in Equation 12. This ODE can be solved analytically:

$$\begin{aligned}\int_0^h \frac{1}{r}dr &= \frac{1}{2}\int_0^h \nabla_T\cdot\vec{u}dt \\ \ln(r_h) - \ln(r_0) &= \frac{1}{2}\int_0^h \nabla_T\cdot\vec{u}dt \\ \ln(r_h) &= \ln(r_0) + \frac{1}{2}(\int_0^h \nabla\cdot\vec{u}dt - \int_0^h \frac{du'}{dx'}dt)\end{aligned}$$

From Equations 3 and 1, the divergence of $\vec{u}$ is:

$$\nabla\cdot\vec{u} = a_1 + b_2 + c_3 \quad (22)$$

and

$$dx' = u'dt \quad (23)$$

Therefore,

$$\begin{aligned}\ln(r_h) &= \ln(r_0) + \frac{1}{2}((a_1 + b_2 + c_3)h - \int_0^h \frac{du'}{u'}) \\ r_h &= r_0\exp(\frac{1}{2}(a_1 + b_2 + c_3)h - \ln(u_h') + \ln(u_0'))\end{aligned}$$

$$r_h = r_0\exp(\frac{1}{2}(a_1 + b_2 + c_3)h)\frac{u_0'}{u_h'} \quad (24)$$

Equation 24 is used to compute the radius of streamtube, where r_h is the streamtube radius at the current time step, r_0 is the radius at the previous time step, u_0' and u_h' are the magnitudes of velocity at the previous step and the current step. Since the magnitude of velocity at current step has been calculated when computing the angle of rotation, there is no unknown value in the right hand side of this equation. The cost of calculating r_h composes only a few multiplications.

3.4 Integration Step Size

The value of h is crucial for integration of particle paths. In [1], Buning suggested to choose this time step size based on the cell size and the inverse of velocity magnitude. Darmofal [2] used a similar method to determine the value of h for tracing particle paths, but for constructing streamribbons, h is furthered restricted by the angle of rotation to produce smoother ribbon surface. In our current implementation, h is fixed for the entire streamline. A default step size is determined for the overall domain by using Buning's method at the preprocessing stage, though h can be interactively modified.

4 Data Structures

To implement the above methods, the major data structures are composed of a list of cell records and a list of node records. To further speed up the construction of streamlines, streamribbons and streamtubes, at the expense of more memory space, we precompute and store the coefficients of the vector field interpolation function, coordinate transformation function, and the specialized Runge-Kutta method during the preprocessing stage.

As a result, a cell record has three coefficient matrices, four node numbers and four cell numbers. The four node numbers are indices of nodes that comprise this tetrahedral cell. The four cell numbers are indices

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE

of cells that are adjacent to this cell. The values stored in a node record include the physical coordinates of the node as well as the vector field on the node. After the preprocessing stage, node records become redundant and can be deleted since the cell records contain all the information needed for performing the particle tracing.

Using our tracing method, each cell record takes

$$(3\ \textit{matrices} + 4\ \textit{node indices} + 4\ \textit{cell indices}) = (3\times(4\times3) + 4 + 4)\times4\ \textit{bytes} = 176\ \textit{bytes}$$

For a typical 500,000-cell data, about 88 megabytes are needed. If memory becomes a problem, the matrices for the interpolation and the transformation functions can be computed on the fly, but the *curl* and *divergence* for each cell then must be stored at the expense of much less memory space, and the memory requirement for each cell becomes 96 bytes.

In order to compare the performance of the new algorithms with the conventional Runge-Kutta methods, we have also implemented the second and the fourth order Runge-Kutta methods. Similarly, to speedup the tracing as much as possible, the matrices for the interpolation and transformation functions are precomputed and stored. Therefore, each cell record takes

$$(2\ \textit{matrices} + 4\ \textit{node indices} + 4\ \textit{cell indices}) = 128\ \textit{bytes}$$

However, the list of node records is needed during the tracing stage. On the other hand, without storing these two matrices, the memory requirement becomes only 32 bytes per cell record, and 24 bytes per node record. To cope with the high memory requirements for visualizing on unstructured grids, some divide-and-conquer strategies must be taken to make possible visualization of large data sets such as those with millions of cells.

5 Test Results

To study the performance of our algorithms, we compare experimentally our specialized Runge-Kutta method (SRK4) with both the conventional second and fourth-order Runge-Kutta methods (RK2 and RK4) for integrating particle paths. To derive fair measurements, as described in previous section, all the needed matrices are precomputed and stored for the implementation of each method. Three data sets are used for our tests. The first data set was generated analytically; it contains 68,921 nodes uniformly positioned in a cubic domain, in which there are totally 320,000 tetrahedra. The vector fields on a node is determined by evaluating three linear functions:

$$\begin{aligned} u_1(x,y,z) &= -0.5x - 6.0y, \\ u_2(x,y,z) &= 6.0x - 0.5y, \\ u_3(x,y,z) &= -2.0z + 20.5. \end{aligned}$$

The second data set is the blunt fin data set obtained from the National Aerodynamic Simulation Facility at the NASA Ames Research Center. This data set was from a computational fluid dynamics simulation of air flow over a flat plate with a blunt fin rising from the plate [5]. 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 originally the computational grid was a single, curvilinear, structured block grid. We converted it into an unstructured grid by splitting each hexahedron into six tetrahedra. The resulting unstructured grid contains 224,874 tetrahedral cells and 40,960 nodes.

We obtained the third data set from the NASA Langley Research Center. It was from a computational fluid dynamics simulation of transonic flow abount an ONERA-M6 wing with free-stream Mach-number 0.84 and 3.06 degrees angle of attack [8]. There are 287,962 tetrahedral cells and 53,961 nodes in this data set.

On each data set, one hundred seed points are randomly selected. Then, streamlines are constructed by using these seed points. The streamline constructions are stopped when either the streamlines reach domain boundaries or the number of time step exceeds a predefined limit (e.g. 1,500).

Since the major function evaluations of all the three methods are of the same kind, i.e. matrix-vector multiplication, we can predict their performances by calculating the number of function evaluations used in these methods. For a single step integration, only one function evaluation is required by using the SRK4 method while four function evaluations are needed if the RK4 method is applied and two function evaluations are performed if the RK2 method is used. Theoretically, the SRK4 method should be faster than the RK2 method by a factor of 2.0, and faster than the RK4 method by a factor of 4.0.

The testing results for the three data sets are shown in Figure 4, Figure 5, and Figure 6. Numbers are seconds and the measurements were performed on a Sun Sparc10 Model 51 (50MHz). Only the core of the integration algorithms was measured. The test results agree with our analysis; the SRK4 method is the fastest method while the RK4 is the slowest one.

The average cost of computing a single step integration by using these three methods are listed in Table 1. Note that now the time unit used is microsecond. According to the timing results listed in Table 1, the speed-up achieved by using the SRK4 method is slightly higher than 2.0 when compared with the RK2 method but may be lower than 4.0 when compared

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE

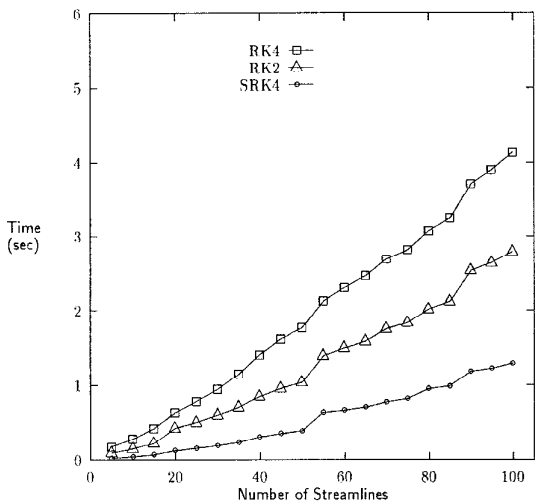

Figure 4: Timing of Constructing Streamlines on Data Set 1

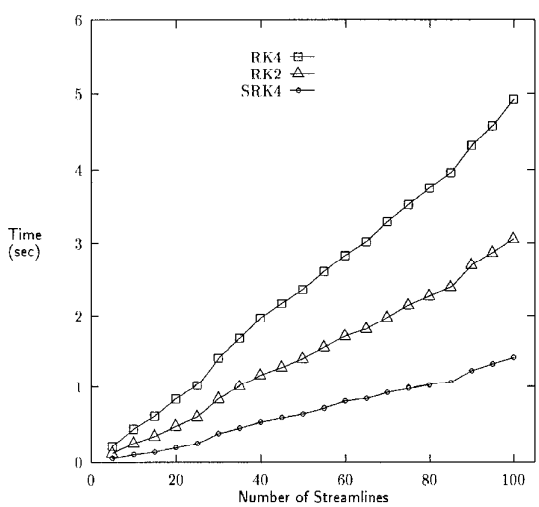

Figure 5: Timing of Constructing Streamlines on Data Set 2

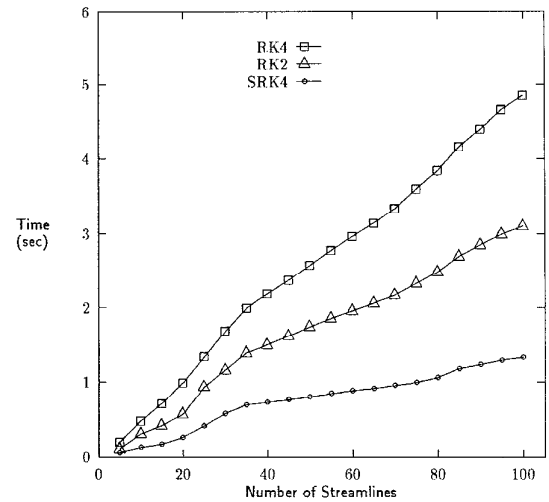

Figure 6: Timing of Constructing Streamlines on Data Set 3

exec. time (in ms)	SRK4	RK2	RK4
Analytical data	10.31	22.20	32.81
Blunt Fin data	9.92	21.33	34.37
ONERA-M6 Wing data	9.00	20.90	32.62

Table 1: Execution Time of a Single Time Step

with the RK4 method. The lower speed-up numbers and the differences between different data sets could be due to both the timing calculations and the overhead for fetching the coefficients of the interpolation functions, etc.

Some visualization results generated by using the algorithms described in this paper are presented in Figure 7. Figure 7 (a) shows a streamribbon image of the analytical data set. From this image, we can see the streamribbons spiral toward a critical point which is a saddle point in the vector field. The streamribbons are colored according to the velocity magnitudes. Figure 7 (b) shows an image of plotting streamtubes in the same data set and using the same initial seed points. This image reveals not only rotation of the flow but also expansion and contraction of the flow.

Figure 7 (c) and (d) show the streamribbon and streamtube visualization of the blunt fin data set. For both images, the view is selected such that the blunt fin is laid down toward the viewer and the plane surface becomes orthogonal to the viewing direction. From these two images, some interesting flow movements are revealed near the leading edge of the fin and the plane.

Figure 7 (e) and (f) display the streamribbon and the streamtube visualization of the ONERA-M6 wing

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE

data set. It is shown in the images that the formation of a wing tip vortex caused by the flow expanding around the wing tip due to pressure differences between the the upper and lower surfaces of the wing.

6 Conclusions

The fourth order Runge-Kutta method is the fundamental procedure for constructing streamlines. A new computational method has been derived to speed up the Runge-Kutta method. A closed form formula is deduced to compute the angular rotation rate of flow for making streamribbons. We have also derived an explicit solution for computing the radius of streamtube that is governed by an ordinary differential equation. The performance of the new methods were measured by using three different data sets on a Sun Sparc10. The test results match our analytic predictions. The speed-up currently achieved can be significant resulting in better interaction when tracing a large number of particles and in a large data space.

While we have improved particle tracing calculations, the use of parallel processing can further speed up the tracing of a significantly large number of particles. In addition, for data sets that do not fit into the main memory of an average workstation, the design of out-of-core or distributed-memory parallel algorithms is needed just to make visualization possible. We are currently designing an out-of-core particle tracing algorithm.

Acknowledgments

This work has been supported in part by the NSF/ACERC and the National Aeronautics and Space Administration under NASA contract NAS1-19480. Thanks go to Dimitri Mavriplis for the Wing data set. Thanks also go to David Darmofal and the anonymous Visualization '95 Conference reviewers who provided many useful suggestions on ways to improve the manuscript.

References

[1] BUNING, P. Sources of Error in the Graphical Analysis of CFD Results. *Journal of Scientific Computing 3*, 2 (1988), 149–164.

[2] DARMOFAL, D., AND HAIMES, R. Visualization of 3-D Vector Fields: Variations on a Stream, January. 1992. AIAA Paper No. 92-0074, AIAA 30th Aerospace Science Meetingand Exhibit.

[3] GOLUB, G. H., AND VAN LOAN, C. F. *Matrix Computations.* The John Hopkins University Press, 1989.

[4] HULTQUIST, J. P. M. Constructing stream surface in steady 3d vector fields. In *Proceeding of Visualization '92* (1992), IEEE Computer Society, pp. 171–178.

[5] HUNG, C.-M., AND BUNING, P. Simulation of Blunt-Fin Induced Shock Wave and Turbulent Boundary Layer Separation, January. 1984. AIAA Paper No. 84-0457, AIAA Aerospace Science Conference.

[6] LOHNER, R., AND AMBROSIANO, J. A Vectorized Particle Tracer for Unstructured Grids. *Journal of Computational Physics 91* (1990), 22–31.

[7] MA, K.-L., AND SMITH, P. Cloud Tracing in Convection-Diffusion Systems. In *Proceeding of Visualization '93 Conference* (October 1993), pp. 253–260.

[8] MAVRIPLIS, D. Unstructured Mesh Algorithms for Aerodynamic Calculations. In *Proceeding of the 13th Int. Conference on Numerical Methods in Fluid Dynamics* (1992), Springer-Verlag, pp. 62–68.

[9] PAGENDARM, H.-G., AND WALTER, B. Feature Detection from Vector Quantities in a Numerically Simulated Hypersonic Flow Field in Combination with Experimental Flow Visualization. In *Proceeding of Visualization '94* (1994), IEEE Computer Society, pp. 117–123.

[10] SCHROEDER, W. J., VOLPE, C. R., AND LORENSEN, W. E. The Stream Polygon: A Technique for 3D Vector Field Visualization. In *Proceeding of Visualization '91* (1991), IEEE Computer Society, pp. 126–132.

[11] VENNARD, J., AND STREET, R. *Elementary Fluid Mechanics.* John Wiley & Sons, Inc., 1975.

[12] VOLPE, G. Streamlines and Streamribbons in Aerodynamics, January 1989. AIAA Paper No. 89-0140, AIAA 27th Aerospace Science Meeting.

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE

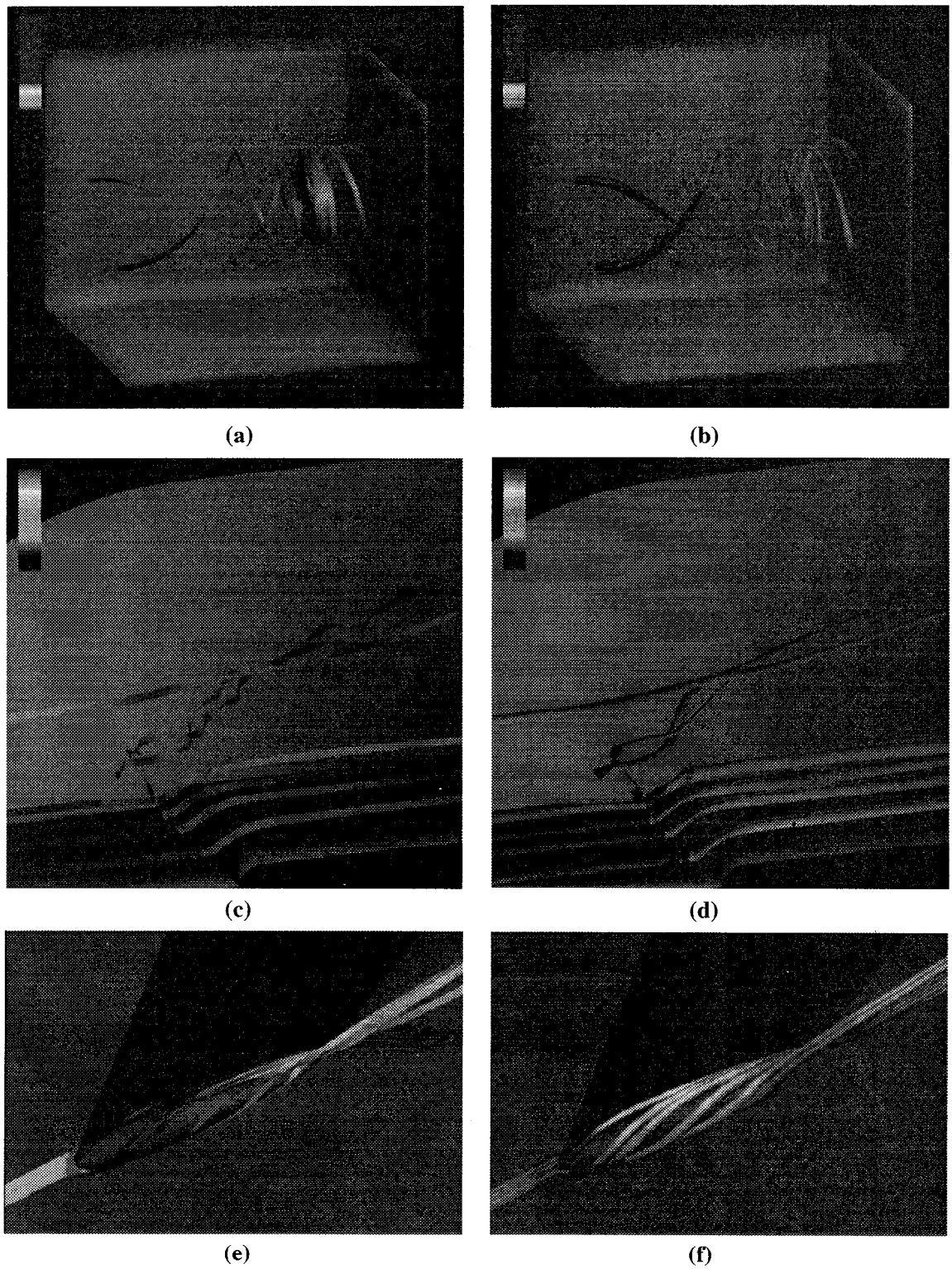

Figure 7: Vector–field Visualization.

Proceedings of the 6th IEEE Visualization Conference (VISUALIZATION '95)
1070-2385/95 $10.00 © 1995 IEEE