Visualization of 3-D Vector Fields : Variations on a Stream

Dave Darmofal and Robert Haimes*

Computational Fluid Dynamics Laboratory

Department of Aeronautics and Astronautics
Massachusetts Institute of Technology, Cambridge, MA 02139

1 Abstract

This paper presents several new algorithms for rendering three-dimensional vector fields. The techniques described are all variations of streamlines. A new algorithm has been developed for streamribbons which avoids many of the difficulties associated with the standard technique of connecting neighboring streamlines. Also, a streamtube algorithm is described where the tube width is based upon the local crossflow divergence. The streamribbon and streamtube ideas are then combined by rendering lines along the surface of the streamtube which rotate according to the local streamwise rotation rate. Finally, a memory-efficient and accurate implementation of unsteady particle paths is presented.

2 Introduction

The visualization of a vector field in three dimensions is a difficult problem. Streamlines are a standard technique by which to visualize steady vector fields. The popularity of streamlines is in part due to their close analogy with common experimental techniques. Historically, in earlier years when compute power and rendering capabilities were more limited, streamlines offered a quick, low memory technique by which to visualize a vector field. Typically, the streamlines would be calculated on a supercomputer and then the small amounts of data generated from this calculation would be brought over to the rendering hardware where line rendering could be quickly performed. Another popular variation of streamlines are streamribbons. Streamribbons have been widely used because they provide information on the local vorticity in a flowfield. Furthermore, streamribbons are often easier than streamlines to understand because of their additional thickness.

In an unsteady flowfield, streamlines and particle paths no longer coincide. Common experimental techniques for unsteady velocity field visualization are streaklines and hydrogen bubbles. A streakline is the line formed by all the particles passing through a given point; in experiments, this technique usually involves the constant injection of a dye at a spot in the flow field. The bubble technique creates hydrogen bubbles by ionization at an electric trip wire. These particles than follow their individual paths. This technique is very useful in boundary layer flows and has been used extensively to show disturbance growth in these situations.

This paper presents several new algorithms useful in visualization of vector fields. All of the algorithms are implemented for unstructured grids in the visualization software package, VISUAL3, developed in the MIT Computational Fluid Dynamics Lab. The streamribbon algorithm described is a more efficient, less troublesome approach to creating ribbons than has been typically used elsewhere. A streamtube algorithm is presented which provides a very useful and intuitive method to visualize the local divergence along a streamline. The streamribbon and streamtube ideas are then combined to simultaneously display the vorticity and the divergence by placing twist lines on the surface of the streamtube. Finally, an efficient and accurate algorithm for unsteady particle path calculation is presented. The results of the variations on steady streamlines are all from the vortical flow calculation over the National Transonic Facility (NTF) delta wing by Becker[?]. The freestream Mach number and Reynolds number for the calculation are 0.85 and 7.5×10^6 respectively. The unsteady results are a channel flow simulation at a turbulent Reynolds number of approximately 180[?].

3 Streamlines

A streamline is a line which is everywhere tangent to the instantaneous velocity field. In a steady flow, a streamline coincides with the path a particle would follow when released from some location. In unsteady flows, streamlines and particle paths are different. The calculation of streamlines amounts to integrating:

$$\frac{d\vec{x}}{d\tau} = \vec{u}(\vec{x}) \tag{1}$$

where $\vec{x}$ is the position in space and $\vec{u}$ is a velocity field. The integration variable, τ, is actually a pseudo-time variable because the integration occurs only at one instant in time.

Since the streamline algorithm is essential to the other algorithms presented, we will now give a thorough explanation of the technique we use to integrate Equation (1)[1].

The method for integrating streamlines is very similar to the techniques used in GRAFIC and PLOT3D, and differs only slightly from the method used by Strid and Eliasson [2]. The algorithm can be broken into two parts, a top-level part which performs the time-integration of the vector function, and a low-level part which is responsible for evaluating the interpolated velocity at an arbitrary point in space.

The top-level task is relatively straightforward. Assuming that somehow one knows the velocity field $\vec{u}(\vec{x})$, then Equation (1) is integrated numerically by a fourth-order Runge-Kutta method. To obtain accurate streamlines at the lowest possible cost, an adaptive time-step is used, with the time-step being halved if the velocity direction changes too much (about 15 degrees) over one timestep, and doubled if it changes sufficiently little (about 2 degrees). The initial time-step is based on the size and the average velocity of the first cell.

The low-level task is to calculate $\vec{u}$ for an arbitrary $\vec{x}$ on an unstructured grid. In VISUAL3, four cell types are currently supported: tetrahedra, pyramids, prisms, and hexahedra. For each cell type there is a mapping from some standard cell of the same type in computational space (ξ_1, ξ_2, ξ_3). The standard cells are illustrated in Figure 1. The mapping can be expressed as

$$\vec{x}(\vec{\xi}) = \sum_j f_j(\vec{\xi})\, \vec{x}_j \tag{2}$$

where the sum is over the nodes of the cell, $\vec{x}_j$ is the coordinate vector of the j^{th} node, and f_j is a function which is equal to 1 at node j and 0 at all other nodes.

For tetrahedra the functions f_j are:

$$\begin{aligned} f_1 &= 1 - \xi_1 - \xi_2 - \xi_3 \\ f_2 &= \xi_1 \\ f_3 &= \xi_2 \\ f_4 &= \xi_3 \end{aligned} \tag{3}$$

For pyramids the functions f_j are:

$$\begin{aligned} f_1 &= \tfrac{1}{4}(1-\xi_1-\xi_3)(1-\xi_2-\xi_3) \\ f_2 &= \tfrac{1}{4}(1+\xi_1-\xi_3)(1-\xi_2-\xi_3) \\ f_3 &= \tfrac{1}{4}(1+\xi_1-\xi_3)(1+\xi_2-\xi_3) \\ f_4 &= \tfrac{1}{4}(1-\xi_1-\xi_3)(1+\xi_2-\xi_3) \\ f_5 &= 1-(1-\xi_3)^2 \end{aligned} \tag{4}$$

For prisms the functions f_j are:

$$\begin{aligned} f_1 &= (1-\xi_1-\xi_2)(1-\xi_3) \\ f_2 &= (1-\xi_1-\xi_2)\,\xi_3 \\ f_3 &= \xi_1\,\xi_3 \\ f_4 &= \xi_1\,(1-\xi_3) \\ f_5 &= \xi_2\,\xi_3 \\ f_6 &= \xi_2\,(1-\xi_3) \end{aligned} \tag{5}$$

For hexahedra the functions f_j are:

$$\begin{aligned} f_1 &= \tfrac{1}{8}(1-\xi_1)(1-\xi_2)(1-\xi_3) \\ f_2 &= \tfrac{1}{8}(1+\xi_1)(1-\xi_2)(1-\xi_3) \\ f_3 &= \tfrac{1}{8}(1+\xi_1)(1+\xi_2)(1-\xi_3) \\ f_4 &= \tfrac{1}{8}(1-\xi_1)(1+\xi_2)(1-\xi_3) \\ f_5 &= \tfrac{1}{8}(1-\xi_1)(1-\xi_2)(1+\xi_3) \\ f_6 &= \tfrac{1}{8}(1+\xi_1)(1-\xi_2)(1+\xi_3) \\ f_7 &= \tfrac{1}{8}(1+\xi_1)(1+\xi_2)(1+\xi_3) \\ f_8 &= \tfrac{1}{8}(1-\xi_1)(1+\xi_2)(1+\xi_3) \end{aligned} \tag{6}$$

Given an arbitrary point $\vec{x}$, and a good guess of the cell in which it is located, a Newton-Raphson procedure is used to calculate the corresponding $\vec{\xi}$. The initial guess $\vec{\xi}^{(0)}$ is taken to be in the center of the cell, and then an improved value is given by

$$\vec{\xi}^{(n+1)} = \vec{\xi}^{(n)} + \left(\frac{\partial \vec{x}}{\partial \vec{\xi}}\right)^{-1} (\vec{x} - \vec{x}^{(n)}) \tag{7}$$

If the correct cell is being searched, this will converge quadratically to the correct point. If the cell being searched does not contain the point, then the new value $\vec{\xi}^{(n+1)}$ will lie outside the standard cell in computational space. The search is then switched to the neighboring cell in the direction of $\vec{\xi}^{(n+1)}$. When doing streamline integrations, the best guess for the cell location is the cell which was used for the last streamline point. In this case, the search seldom has to move through more than one or two cells.

Having found the correct cell and the value for $\vec{\xi}$, the velocity is given by

$$\vec{u} = \sum_j f_j(\vec{\xi})\, \vec{u}_j \tag{8}$$

To spawn a streamline, VISUAL3 uses a unique cursor pointing ability[3]. The user positions the cursor on the two-dimensional display of a cutting plane and uses a mouse button click to start a streamline at that location. By controlling the cutting plane and the cursor position, a streamline may be quickly (and intuitively) spawned at any location in the computational domain.

Figure 2 shows three streamlines rendered with the color a function of the local Mach number. Two streamlines are in the primary vortex: one at the vortex axis, the other in the outer region of the vortex. The third streamline travels underneath the primary vortex and is eventually entrained in the secondary vortex. A difficulty with the standard streamline technique is that spatial relationships are difficult to determine because of the lack of shading and thickness cues.

4 Streamribbons

Streamribbons are streamlines that have been given some width and rotation based on the local streamwise vorticity. Many implementations of streamribbons calculate two streamlines with the second line constrained to remain a constant distance away from the original line[4][5][6]. In a highly sheared flow, neighboring streamlines may have large velocity differences causing the slower streamline to require many more time steps before completing the integration. Thus, the construction of triangles forming the ribbon surface will be extremely difficult in a shear flow because the slower streamline will contain many more points. The end result is the need for a complicated algorithm to build ribbon surfaces. The approach here is slightly different. One edge is the true streamline, the other edge is contructed by rotating a constant length normal vector about the line according to the local streamwise angular rotation rate[7]. The result is a ribbon whose twist reflects the streamwise curl of the vector field of the flow. The equation which describes the rotation of the streamline normal is:

$$\begin{aligned} \frac{d\theta}{d\tau} &= \frac{1}{2}(\vec{\omega} \cdot \vec{s}) \\ \vec{s} &= \frac{\vec{u}}{|\vec{u}|} \\ \vec{\omega} &= \nabla \times \vec{u} \end{aligned} \tag{9}$$

where θ is the current angle of the normal with respect to its initial direction. The factor of $\frac{1}{2}$ is needed because the angular rotation rate is half the curl of the vector. Equations (1) and (9) are then integrated simultaneously. The curl of the velocity vector field is evaluated numerically using the same parametric representations in computational space as used for the streamline algorithm.

$$\begin{aligned} & \frac{\partial \vec{u}}{\partial \vec{\xi}} &=& \frac{\partial \vec{u}}{\partial \vec{x}} \frac{\partial \vec{x}}{\partial \vec{\xi}} \\ \Longrightarrow \quad & \frac{\partial \vec{u}}{\partial \vec{x}} &=& \frac{\partial \vec{u}}{\partial \vec{\xi}} \left(\frac{\partial \vec{x}}{\partial \vec{\xi}} \right)^{-1} \\ & &=& \left(\sum_j \vec{u}_j \frac{\partial f_j}{\partial \vec{\xi}} \right) \left(\sum_j \vec{x}_j \frac{\partial f_j}{\partial \vec{\xi}} \right)^{-1} \end{aligned} \tag{10}$$

A possible interpretation difficulty arises because of the 360 degree periodicity of the current normal angle. To eliminate this difficulty, consecutive normal angles are forced to be at most 30 degrees different. If a separation of greater than 30 degrees occurs, the pseudo-time step is simply decreased and the current integration step is re-calculated. This constraint on the change of angle also results in the rendering of smoother ribbons.

The final step is choosing a normal direction for the first node of the streamline. Initially, this is done automatically; however, the user may rotate this position interactively and thus rotate the entire ribbon. This rotation feature is extremely helpful in determining exactly how and which direction a ribbon twists. Construction of the ribbon surface into polytriangle strips can be done straightforwardly by simply joining a streamline node to its normal node and then connecting this to the next streamline node. In this manner, the entire ribbon becomes a single polytriangle strip (see Figure 3). Furthermore, this method of ribbon calculation avoids the problem of constraining a second streamline and is extremely easy to implement in comparison.

Figure 4 has the streamribbon results from the same streamlines as Figure 2. The high rotation rate of the primary vortex axis streamline can be clearly seen. The other primary vortex streamribbon shows surprisingly little twist as it travels around the vortex axis. This lack of twist may be an indication that the vortex core is in solid body rotation while the outer region is an irrotational flow. Some twist can be seen in the remaining ribbon but the exact details are hard to recognize from the single view.

5 Streamtubes

The precise definition of a streamtube is the surface formed by all the streamlines passing through a closed contour[8]. The construction of a streamtube involves the same difficulties as the construction of a streamribbon. To avoid these problems, a slightly different approach is used. The streamtube is considered to have a circular crossflow area with the radius derived from the local crossflow divergence. The crossflow divergence measures the local crossflow expansion rate. Specifically, we have:

$$\nabla_T \cdot \vec{u} = \nabla \cdot \vec{u} - \frac{du'}{dx'}$$

where $\nabla_T \cdot \vec{u}$ is the crossflow or transverse divergence and the primed derivative represents the change of the x' velocity component with the x'-axis aligned with the local velocity vector. The equation governing the expansion of the flow in the transverse direction is:

$$\frac{1}{r}\frac{dr}{d\tau} = \frac{1}{2}\nabla_T \cdot \vec{u}$$

which may be rewritten as:

$$r = r_0 \exp \left\{ \frac{1}{2} \int_{\tau_0}^{\tau} \nabla_T \cdot \vec{u} \, d\tau' \right\} \tag{11}$$

Thus, r, the crossflow radius, is an exponential function of the streamwise integral of the crossflow divergence. The integral in Equation (11) is integrated forward in pseudo-time along with Equation (1). The initial crossflow radius, r_0, is controlled by the user.

To render the tube, each circular crossflow section is divided into equal arcs. We have found that 30 degree arcs provide adequately smooth tubes. For most streamtubes, rendering poly-triangle strips in the streamwise directions results in longer strips than if crossflow sections were rendered; thus, rendering tubes in the streamwise direction will increase the algorithm performance. A user-set radius limiter is also needed to ensure the width of a streamtube does not become so large so as to obscure an entire image. This rapid growth in the streamtube width is especially difficult near stagnation points where the radius becomes extremely large.

Figure 5 shows the streamlines rendered using the streamtube technique. The three-dimensional relations of the various streamlines can be clearly seen from both shading and thickness cues. Furthermore, the rapid expansion of the secondary vortex streamtube is readily detectable. This region is associated with vortex breakdown at the trailing edge. The color and the size of the tube both indicate that the flow is stagnating.

6 Streamtubes with twist

The rotation and divergence effects can be rendered simultaneously by placing lines on the surface of the streamtube which twist with the local rotation rate. The only additional difficulty which arises in rendering the twist lines is that the lines must lie on a facet between poly-triangle strips. This guarantees that the lines will lie on top of the streamtube surface. However, to place the twist lines on a facet requires that the poly-triangle strips must also twist as they travel downstream. Thus, the entire rendering algorithm consists of:

- Render twisted streamwise poly-triangle strips
- Render twisted lines along facets of poly-triangle strips

The final image displays the streamline, the rotation rate, the crossflow divergence, and scalar variations.

Figure 6 shows the streamtubes with twist. Again, we receive the same information regarding the streamtube expansion as with streamtubes. In addition, the streamwise rotation is now visible. The twist lines are much more effective than streamribbons at conveying the twist information. The lack of rotation of the outer primary vortex streamline is quite clear. Also, in the secondary vortex, the streamwise vorticity undergoes a sudden change. Beyond the rapid tube expansion region, the twist no longer exists and confirms that the secondary vortex has undergone a breakdown.

7 Unsteady Particle Paths

The final variation of streamlines is unsteady particle paths. This involves an integration in time (unlike the streamlines which are in pseudo-time). Specifically, the position of a particle is given by:

$$\frac{d\vec{x}}{dt} = \vec{u}(\vec{x}, t) \tag{12}$$

In choosing a numerical scheme to integrate Equation (12), three important factors must be considered:

1. The velocity vector field, $\vec{u}$ is changing in time. If the numerical scheme requires several time planes of the velocity field, either all time planes will remain in memory or each time plane will be read from disk when needed. Thus, for large data sets, this type of scheme will place extreme demands either on memory usage or memory bandwidth. Thus, optimally, the integration should require only one time plane to be active in memory.
2. The integration should have good accuracy to make certain the numerical particle paths are a reasonable representation of the true paths. Thus, simple schemes such as forward or backward Euler, although requiring only one time plane of the velocity field will be inadequate. Furthermore, adaptive time stepping cannot be performed because the time step between new positions is set by the time step of the user's data.
3. Since the time step is not determined by VISUAL3, the numerical technique must be stable for all step sizes. In otherwords, if the local linearized problem predicts an exponentially decaying change in position, then the numerical scheme should also have a decaying solution. If the local linearized problem has an oscillatory solution then the numerical scheme should at least not predict an exponentially growing solution.

With these three factors in mind, a good choice for the integration scheme is backwards differentiation. The general form of an order s accurate backwards differencing formula is:

$$\sum_{i=1}^{s} \frac{1}{i} \nabla^i \vec{x}^{n+1} = k\, \vec{u}^{n+1} \tag{13}$$

where k is the time step and ∇ is the usual backwards difference operator:

$$\nabla \vec{x}^{n+1} = \vec{x}^{n+1} - \vec{x}^{n}$$

This implicit scheme is stable up to the 6th order formula. We use the fourth order scheme which matches the accuracy of the Runge-Kutta integration used for the streamline algorithm. Only the new time plane of the velocity vector field is needed for the integration. The calculation does require the storage of the s previous positions but this is simply 3 floating points per path per time step and this information is generally kept for rendering purposes.

The fourth-order accurate version of Equation (13) can be re-written as

$$\begin{aligned}(1+\tfrac{1}{2}+\tfrac{1}{3}+\tfrac{1}{4})\, \vec{x}(\vec{\xi}^{(n+1)}) - k\, \vec{u}(\vec{\xi}^{(n+1)}) = \\ \text{function}(\vec{x}^{(n)}, \vec{x}^{(n-1)}, \vec{x}^{(n-2)}, \vec{x}^{(n-3)}).\end{aligned}$$

This equation is nonlinear in $\vec{\xi}^{(n+1)}$ and is solved by a minor variation of the Newton-Raphson algorithm used in the streamline calculation to determine the computational coordinates $\vec{\xi}$.

The calculation and display of unsteady particle paths occurs in real time. A particle path may be started in two ways:

1. A particle is released at every time step from the current cursor position. This simulates the experimental streakline technique.
2. A set of equally-spaced particles is released from a line or circle and are tracked at every time step. This simulates the experimental hydrogen bubble technique.

Spheres are used to indicate the position of the particles. The current and the three previous particle positions (which are stored for the numerical integration) may be rendered. VISUAL3 does not require a sphere graphic primitive for rendering the particles; instead, a icosahedron is used to approximate a sphere.

Figure 7 is a hydrogen bubble simulation using the unsteady particle path algorithm described above. The bubble color represents the velocity perturbation (i.e., the magnitude of the instantaneous velocity minus the local mean velocity). The real time bubble simulation was performed on a subset of the results from the channel flow calculation and contained approximately 250,000 nodes. On this size grid, each integration in time of the entire set of bubble paths takes about 1 second on a Stardent GS-2000; thus, the real time animation is quite reasonable.

8 Conclusions

The new techniques presented here have proven extremely useful in the visualization of 3-D vector fields. A new streamribbon algorithm is described which eliminates the difficulty of connecting diverging streamlines. A novel streamtube technique provides an intuitive tool for the display of local divergence effects. Furthermore, the additional visual effects from shading and thickness make the streamtube method more readily understandable than streamlines. The tubes with twist provide even more insight by displaying the local streamwise vorticity. Finally, a new unsteady particle path scheme has been implemented which is memory efficient and accurate.

Acknowledgements

The authors wish to thank two people without whom this work would not be possible: Professor Mike Giles and Professor Earll Murman. Their assistance and ideas have been invaluable to this paper. Dave Darmofal would like to thank the DOD NDSEG Fellow program for their support of his research and education. This work was partially supported by AFOSR under grant 89-0395 monitored by Dr. Len Sakell.

References

[1] M. Giles and R. Haimes. Advanced Interactive Visualization for CFD. *Computing Systems in Engineering*, 1(1):51–62, 1990.

[2] T. Strid and A. Rizzi. Development and use of some flow visualization algorithms. VKI Lecture Series on Computer Graphics and Flow Visualization in CFD, 1989.

[3] R. Haimes and D. Darmofal. Visualization in Computational Fluid Dynamics: A Case Study. IEEE Computer Society, Visualization '91, October 1991.

[4] G. Volpe and R. E. Melnik. The design of transonic airfoils by a well-posed inverse method. Proceedings, International Conference on Inverse Design Concepts in Engineering Sciences, October 1984.

[5] J.P.M Hultquist. Interactive Numerical Flow Visualization Using Stream Surfaces. *Computing Systems in Engineering*, 1(2-4):349–353, 1990.

[6] M. Siclari, M.J. Mandel and A. Scuderi. Flow Visualization Research at Grumman's Computational Aerodynamics Laboratory, 1989.

[7] D. Darmofal. Hierarchal Visualization of Three-Dimensional Vortical Flow Calculations. Master's thesis, M.I.T., March 1991.

[8] G. K. Batchelor. *An Introduction to Fluid Dynamics*. Cambridge University Press, 1967.

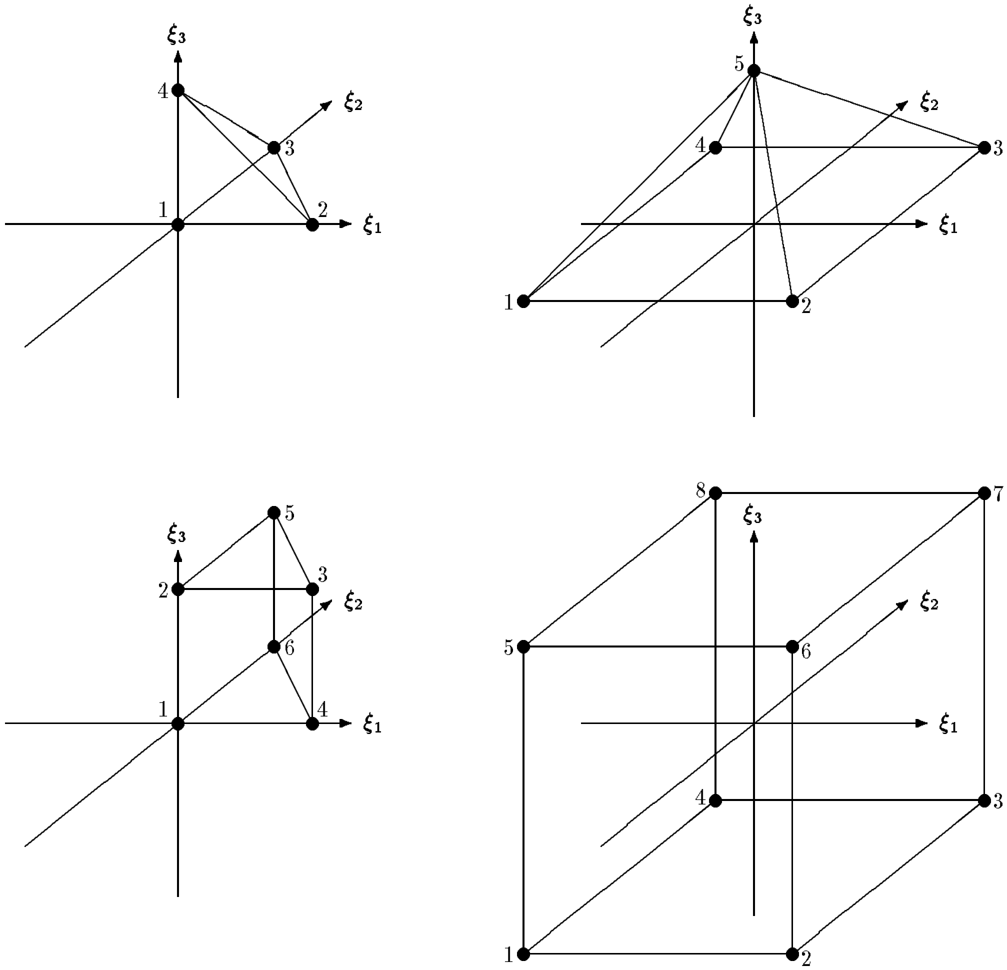

Figure 1: Four standard cells in computational space

Figure 2: Streamlines

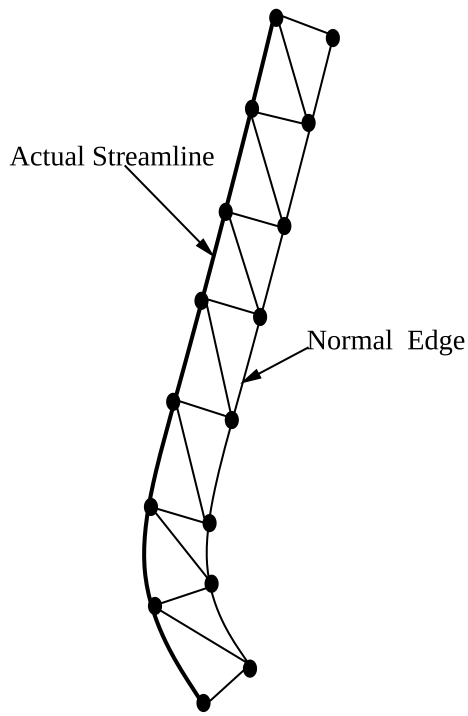

Figure 3: Polytriangle strip for a streamribbon

Figure 4: Streamribbons

Figure 5: Streamtubes

Figure 6: Streamtubes with twist

Figure 7: Unsteady particle paths