

Scientific Visualization  
Overviews Methodologies  
Techniques

G. Nielson, H. Müller, and H. Hagen, editors



## Chapter 5

# Scientific Visualization of Large-Scale Unsteady Fluid Flows

David A. Lane

**Abstract.** *In a numerical flow simulation, it is common to generate several thousand time steps of unsteady (time-dependent) flow data. Each time step may require tens to hundreds of megabytes of disk storage, and the entire unsteady flow data set may be hundreds of gigabytes. Interactive visualization of unsteady flow data of this magnitude is impossible with the current hardware capabilities. Particle tracing is an effective technique to visualize unsteady flows. Streaklines, which are computed by tracking particles in unsteady flows, depict time-varying phenomena that are sometimes difficult or impossible to see with other flow visualization techniques. This chapter provides a tutorial for particle tracing in steady and unsteady flows. First, the life cycle of a typical numerical flow simulation is outlined. The current approaches for visualizing unsteady flows are then described. Many systems exist for flow visualization, some of which are discussed. A tutorial for particle tracing is then given. A particle tracing system called UFAT has been developed for unsteady flow visualization. The features and implementation of UFAT are described. The steps for computing streaklines are summarized. Several unsteady flow data sets were visualized using UFAT, and the results are presented.*

### 5.1 Introduction

The life cycle of a numerical flow simulation in Computational Fluid Dynamics (CFD) generally consists of three phases: grid generation, flow calculation, and visualization. First, a numerical grid is constructed to enclose the boundaries of the object in the flow using a grid generation tool. The grid may be rectilinear, curvilinear (structured), block structured, unstructured, or a mix of the structured and unstructured (hybrid). In CFD, structured and unstructured grids are commonly used. A multizoned curvilinear grid consists of one or

more structured grids (sometimes referred to as blocks or zones). Some blocks may overlap other blocks. Each cell in a 3D curvilinear grid has a hexahedron-like configuration (warped bricks); the faces of the cell will not generally be planar. Figure 5.1 shows a multizoned curvilinear grid. An unstructured grid consists of cells that are not necessarily hexahedra. For example, the cells may be tetrahedra, prisms, or pyramids. For a good discussion of the types of grids used in scientific visualization, see Speray and Kennon [29]. In the second phase of the flow simulation, a system of Navier-Stokes equations that simulate the flow condition is solved. Euler equations are commonly used. The solution yields momentum and other flow quantities such as density and stagnation energy. From these quantities, velocity can be derived. Several methods can be used to set up the flow equations. Using a finite difference method, the flow quantities are solved at the grid points. A finite volume method solves the flow quantities, usually at the grid cell centers instead of at the grid points. Using a finite element method, the flow quantities in each cell are represented by a set of basis functions. The calculation of the flow solution can be computationally expensive, depending on the grid size. In the last phase, the velocity field and other flow quantities computed from the flow calculation are analyzed using a number of visualization techniques, including contour plots, particle traces, isosurfaces, and volume rendering.

With the increase in computing power over the past decade, numerical simulations of 3D unsteady flow fields are becoming more common. The disk requirements for 3D unsteady flow simulations have also increased dramatically. The data generated from an unsteady flow simulation is usually several orders of magnitude larger than the data generated from a steady flow simulation. Some unsteady flow data sets require hundreds of gigabytes of disk space. Visualizing data of this magnitude is a Grand Challenge problem in scientific visualization.

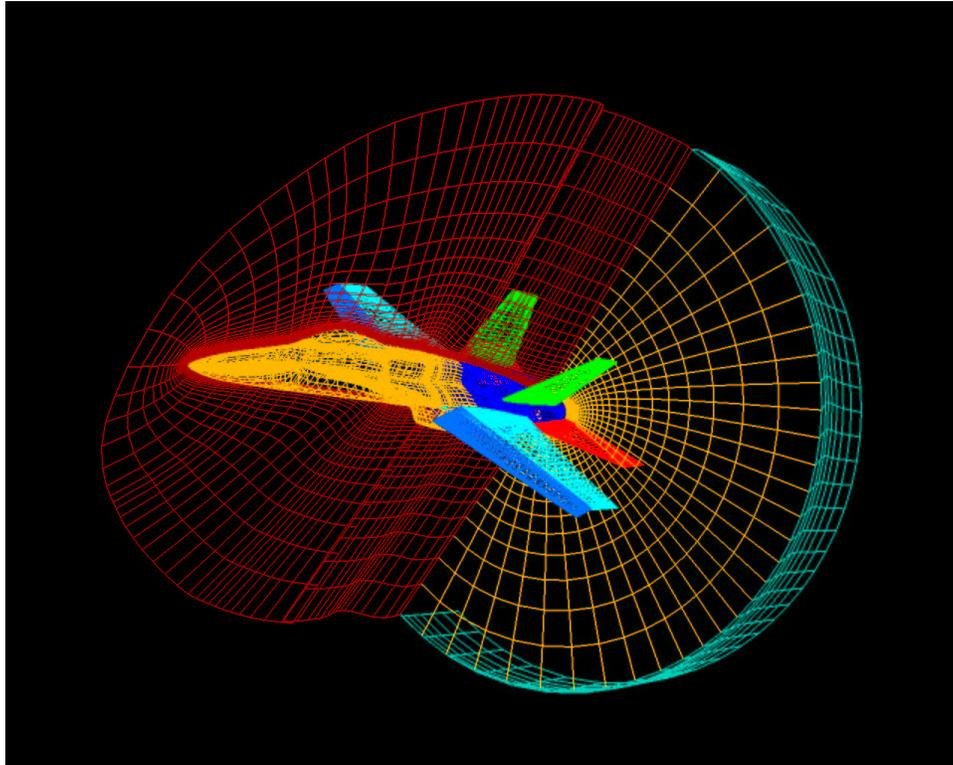
## 5.2 Problem

At NASA Ames Research Center, CFD scientists are performing complex 3D unsteady flow simulations using the supercomputers located at the Numerical Aerodynamic Simulation (NAS) facility. Table 5.1 shows five grids that were recently used in several simulations.

Grid	No. of Grid Points	No. of Blocks
Clipped Delta Wing	$250 \times 10^3$	7
Descending Delta Wing	$900 \times 10^3$	4
V-22 Tilt Rotor	$1.3 \times 10^6$	25
Harrier Jet	$2.8 \times 10^6$	18
SOFIA Airplane	$3.2 \times 10^6$	35

**Table 5.1:** Five multizoned curvilinear grids used for unsteady flow simulation.

Two types of files are usually generated in a simulation: the grid file contains the physical coordinates of the grid points and the solution file contains the momentum and other



**Figure 5.1:** A multizoned curvilinear grid surrounding a F18 aircraft, generated by Yehia Rizk and Ken Gee, NASA Ames Research Center. Each sub-block is distinguished by color.

flow quantities computed in the flow calculation. The size of the grid file depends on the number of grid points and the size of the solution file depends on the flow quantities saved from the flow calculation. If the grid consists of multiple blocks, then the grid file also contains an integer code for each grid point that indicates if the point is in an overlapped region or near a wall boundary (see “Particle Tracing in Multizonal Grids” below). In unsteady flows, there is one solution file per time step. If the grid moves in time, then there is also one grid file per time step. Table 5.2 shows the disk requirements for storing the grid and solution files per time step for the grids given in Table 5.1. In Table 5.2, the solution files assume that five flow quantities are saved: stagnation energy, density, and the  $x, y, z$  components of momentum.

It is common to have thousands of time steps in an unsteady flow simulation. Table 5.3 shows the disk requirement for storing the data generated in one flow simulation for each of the data sets shown in Table 5.2. For example, the descending delta wing consists of 90,000 time steps. The disk space required to save all of these time steps is 3,240 GB ( $90,000 \times 36$  MB). Clearly, it is impossible to interactively visualize data of this magnitude with the current hardware capability.

Grid	Grid File	Solution File	Total Per Time Step
Clipped Delta Wing	4 MB	5 MB	9 MB
Descending Delta Wing	16 MB	20 MB	36 MB
V-22 Tilt Rotor	23 MB	29 MB	52 MB
Harrier Jet	45 MB	56 MB	101 MB
SOFIA Airplane	53 MB	66 MB	119 MB

**Table 5.2:** The disk space, in megabytes (MB), required for each time step.

Grid	Number of Time Steps	Total Per Simulation
Clipped Delta Wing	5,000 per cycle for 3 cycles	135 GB
Descending Delta Wing	90,000	3,240 GB
V-22 Tilt Rotor	1,450 per cycle for 3 cycles	226 GB
Harrier Jet	1,000	101 GB
SOFIA Airplane	10,000	1,190 GB

**Table 5.3:** The disk space, in gigabytes (GB), required per flow simulation.

### 5.3 Current Approaches

For interactive visualization, it is ideal to store all time steps of the data in the physical memory of the system so that the scientist can loop through the time steps interactively. However, as shown in Table 5.3, some 3D unsteady flow data sets are too large to be visualized interactively. Even the simple task of storing a few time steps of the flow data becomes a problem due to insufficient disk space. Therefore, postvisualization is necessary. For postvisualization, scientists often save their data during the flow calculation at fixed time intervals. The flow data set that is extracted this way will be referred to as the *saved data set* hereafter. For example, every 50th time step of the descending delta wing data (see Table 5.2) was saved for postvisualization. The disk requirement for the saved data set is 64.8 GB ( $90,000/50 \times 36$  MB). Currently, it is difficult to find a system with tens of gigabytes of memory for interactive visualization.

Two approaches for postvisualization are sometimes used. The first approach is to store as many time steps of the saved data as possible in memory. If the size of the flow data is small (that is, on the order of hundreds of megabytes), then this approach is attractive. However, for large-scale data sets such as the V-22 and SOFIA data sets listed in Table 5.3, this approach would allow only a few time steps to be visualized interactively. This is unacceptable for most simulations. The second approach is to subsample the flow data at a lower grid resolution so that more time steps can fit in memory. However, this is generally not a good approach because the resolution of the flow is lost. For example, the presence of a vortex may not be detected. Furthermore, additional preprocessing time is required for subsampling. Both of these two approaches only allow the scientist to visualize a subset of the saved data, and important characteristics of the flow may not be displayed.

A more effective approach is to use every time step of the saved data without subsam-

pling. A requirement is that the system has enough memory to store at least a few time steps of the data. The flow data are visualized by loading one time step of data into memory and then applying the desired visualization techniques one time step at a time. For example, compute color contours on a grid surface at each time step for all time steps. Visualizing flow data at instants in time is sometimes referred to as *instantaneous flow visualization*. A good survey of several instantaneous flow visualization techniques can be found in Post and van Wijk [23].

The interactive rate of instantaneous flow visualization is limited by the disk I/O performance of the system. If the flow data at each time step is hundreds of megabytes, then the time required to read each time step's data could be a few seconds, depending on the system's disk I/O performance. For example, the Harrier jet requires 101 MB per time step for both the grid and solution data files (see Table 5.2). It would take approximately one to two seconds to read the grid and solution files at each time step if the disk I/O rate ranges from 50 MB to 100 MB per second. In addition to the read time, computation time is required by the visualization technique used; for example, particle tracing and isosurface calculation. If there are thousands of particles, then it may take several more seconds or minutes for the computation. For this reason, the interactive rate of instantaneous flow visualization of large-scale flow data can be very slow. Although instantaneous flow visualization can be effective (though slow) for some analysis, it may not reveal time-varying phenomena in the flow. Effective unsteady flow visualization techniques should consider time as a parameter in the calculation to depict flow phenomena that evolve in time. Time-dependent particle tracing, which is described in Section 5.5, is an example of an unsteady flow visualization technique.

It is also possible to incorporate visualization into the flow calculation phase. This method generates the graphics objects (for example, particle traces) while the solutions are being computed. The method would also reduce disk requirements because the solution files do not need to be saved for postvisualization; however, when visualization parameters (for example, seed locations) need to be changed, the flow would need to be recalculated, which is computationally expensive.

## 5.4 Flow Visualization Systems

There are many existing systems that allow instantaneous flow visualization. PLOT3D (Buning and Steger [6]) is a popular CFD visualization tool that provides several instantaneous visualization techniques. FAST (Bancroft et al. [1]) consists of a set of modules that support PLOT3D functionalities and several other new features like topology, function calculator, and animation. COMADI (Vollmers [31]), HIGHEND (Pagendarm [21]), Visual3 (Haimes and Giles [12]), and many other visualization systems also provide several instantaneous visualization techniques. Most of these systems allow unsteady flow visualization by looping through the time steps and visualizing the flow data one time step at a time; however, many of them do not consider time as a parameter in the calculation.

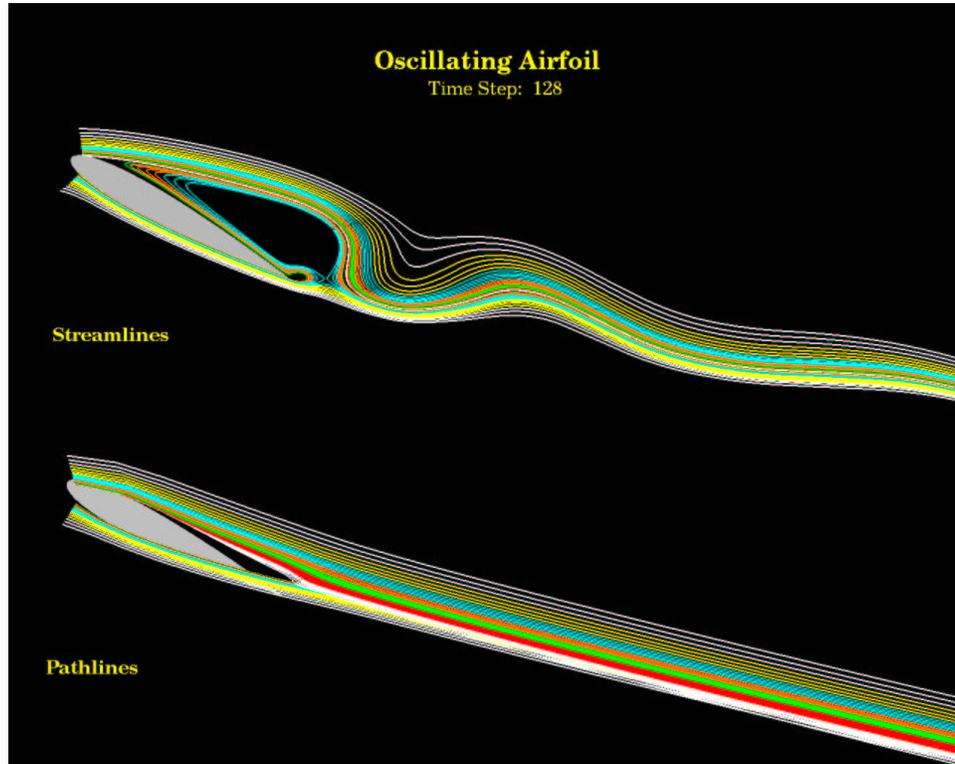
Because simulations of large-scale 3D unsteady flows have only become possible within the past decade, there are currently few systems developed specifically for unsteady flow visualization. PLOT4D and Streaker developed by Smith et al. [28] perform unsteady flow visualization by visualizing trivariate functions of two spatial variables and one time

variable. For example,  $f(x, y, t)$  or  $f(x, z, t)$ . Jespersen and Levit [14] and Vaziri et al. [30] have developed unsteady flow visualization systems using the CM5 for the calculation and the results are sent over the network to a local graphics workstation for display. The Virtual Wind Tunnel (Bryson and Levit [3]) performs distributed unsteady flow visualization using a Convex C3240 system and an SGI graphics workstation. The particle traces are computed on the Convex system and then sent over the network to the graphics workstation which displays the traces and provides local view manipulation. pV3 (Haines [11]) distributes the computation to one or more servers using the PVM message-passing library. A nice feature of pV3 is that it allows the user to view the solution while it is being computed. This “plug-into/unplug” feature allows the user to monitor the flow calculation and to terminate the calculation when necessary. Only a few of the systems mentioned above compute streaklines, pathlines, and timelines.

I have developed a particle tracing system called the Unsteady Flow Analysis Toolkit (UFAT) which computes particle traces in unsteady flow [17, 18]. UFAT is unique from existing systems in that it computes particle traces from a large number of time steps, performs particle tracing in unsteady flow with moving curvilinear grids, supports a save/restore option, and allows playback (see Section 5.6).

## 5.5 Particle Tracing

An effective way to visualize unsteady flow is to compute particle traces. In steady flows, a *streamline* is a field line tangent to the velocity field at an instant in time. For unsteady flows, three types of particle traces can be computed: pathlines, streaklines, and timelines. A *pathline* shows the trajectory of a single particle released from one fixed location, called the *seed location*. A *streakline* is a line joining the positions, at an instant in time, of all particles that have been previously released from a seed location. Streaklines can be simulated by releasing particles continuously from the seed locations at each time step. In hydrodynamics, streaklines can be simulated by releasing small hydrogen bubbles rapidly from the seed locations. For streakline calculation, particles are released from specified seed locations and tracked through the given time steps. A *timeline* is a line connecting particles that have been released at the same time. In an instantaneous flow field, streamlines, pathlines, and streaklines are identical (Schlichting [26]). In experimental flow visualization, timelines are simulated by releasing a line of particles simultaneously at some time interval. Figures 5.2 and 5.3 show streamlines, pathlines, streaklines, and timelines computed near an oscillating 2D airfoil. Streaklines can reveal very different information than those shown with instantaneous streamlines. The vortices behind the airfoil are visible with the streaklines shown in Figure 5.3, but the streamlines shown in Figure 5.2 do not reveal them clearly. Furthermore, animated streaklines effectively show the development of flow phenomena in time. There are many research papers on the subject of particle tracing in steady flows; however, few of them discuss time-dependent particle tracing. This section provides a tutorial for particle tracing in steady and unsteady flows.



**Figure 5.2:** Streamlines and pathlines pass through an oscillating airfoil. Particle traces are colored by release location.

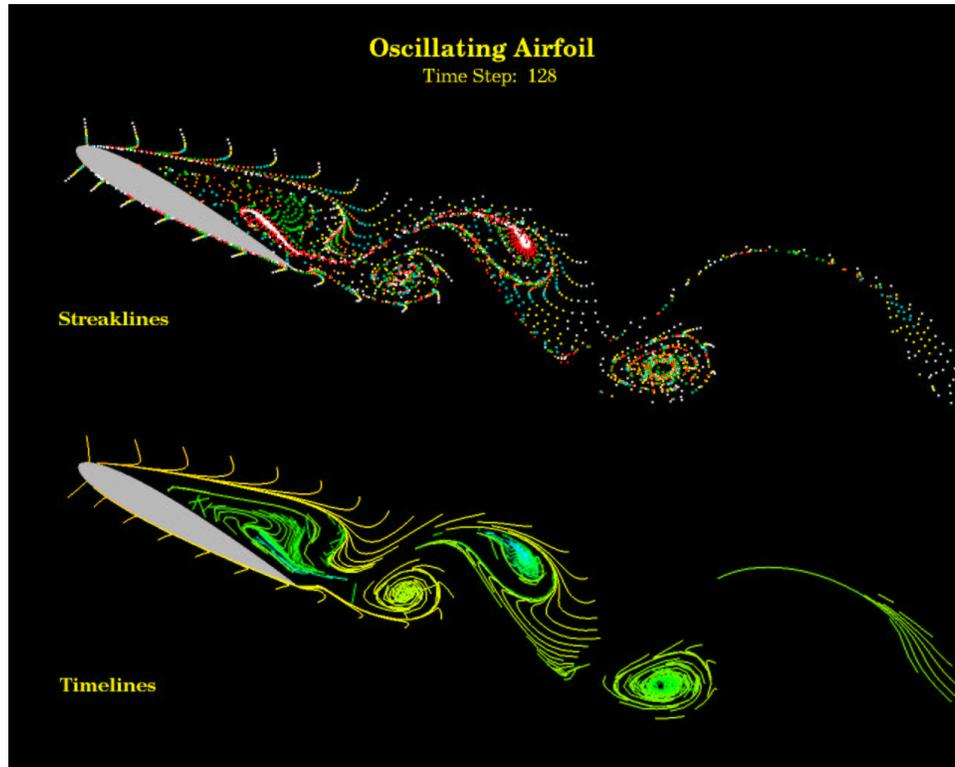
### 5.5.1 Numerical Models

Most existing methods for particle tracing are based on numerical integration schemes. There are implicit methods for computing streamlines using stream functions (see Kenwright and Mallinson [16]); however, most of these methods have not been extended for unsteady flows. In the next section, a multistage method based on the fourth-order Runge-Kutta (RK4) integration is described for particle tracing in steady flows. In the following section, the method is then extended for unsteady flows.

#### Particle Tracing in Steady Flows

Given a vector function  $\mathbf{v}(\mathbf{p})$  defined for all  $\mathbf{p}$  in the grid domain  $G$ , the path of a massless particle at position  $\mathbf{p}$  can be determined by solving the following ordinary differential equation:

$$\frac{d\mathbf{p}}{dt} = \mathbf{v}(\mathbf{p}(t)). \quad (5.1)$$



**Figure 5.3:** Streaklines and timelines pass through an oscillating airfoil. Streaklines are colored by release location and timelines are colored by release time.

Function  $\mathbf{p}(t)$ , which represents the position of the particle at time  $t$ , can be computed by integrating (5.1). Hence, the particle position after time  $\Delta t$  is as follows:

$$\mathbf{p}(t + \Delta t) = \mathbf{p}(t) + \int_t^{t+\Delta t} \mathbf{v}(\mathbf{p}(t)) dt. \quad (5.2)$$

Equation (5.2) can be evaluated using a numerical integration scheme. A common scheme is the RK4 method. Let  $k = 0$  and  $\mathbf{p}_k$  be the current position of the particle. Then, for  $\mathbf{p}_k \in G$ , let

$$\begin{aligned} \mathbf{a} &= h\mathbf{v}(\mathbf{p}_k), \quad \mathbf{b} = h\mathbf{v}(\mathbf{p}_k + \mathbf{a}/2), \quad \mathbf{c} = h\mathbf{v}(\mathbf{p}_k + \mathbf{b}/2), \quad \mathbf{d} = h\mathbf{v}(\mathbf{p}_k + \mathbf{c}), \\ \mathbf{p}_{k+1} &= \mathbf{p}_k + (\mathbf{a} + 2\mathbf{b} + 2\mathbf{c} + \mathbf{d})/6, \quad \text{and} \quad k = k + 1, \end{aligned} \quad (5.3)$$

where  $h = \Delta t$  is the step size. To obtain the path of the particle at  $\mathbf{p}_k$ , (5.3) is evaluated repeatedly until  $\mathbf{p}_k$  leaves the grid domain  $G$ .

### Particle Tracing in Unsteady Flows

In unsteady flows, velocity changes in time, so  $\mathbf{v}$  is a function of space and time. Suppose the vector function  $\mathbf{v}(\mathbf{p}, t)$  is defined for all  $\mathbf{p} \in G$  and  $t \in [t_1, t_n]$ , where  $n$  is the number of time steps in the unsteady flow. The path of a particle at position  $\mathbf{p}$  can be computed by solving the following equation:

$$\frac{d\mathbf{p}}{dt} = \mathbf{v}(\mathbf{p}(t), t). \quad (5.4)$$

Integrating (5.4) yields:

$$\mathbf{p}(t + \Delta t) = \mathbf{p}(t) + \int_t^{t+\Delta t} \mathbf{v}(\mathbf{p}(t), t) dt. \quad (5.5)$$

As with steady flows, we can integrate (5.5) numerically using the RK4 method. Let  $t = t_1$ ,  $k = 0$ , and  $\mathbf{p}_k$  be the current position of the particle. Then, for  $t < t_n$  and  $\mathbf{p}_k \in G$ , let

$$\begin{aligned} \mathbf{a} &= h\mathbf{v}(\mathbf{p}_k, t), & \mathbf{b} &= h\mathbf{v}(\mathbf{p}_k + \mathbf{a}/2, t + h/2), \\ \mathbf{c} &= h\mathbf{v}(\mathbf{p}_k + \mathbf{b}/2, t + h/2), & \mathbf{d} &= h\mathbf{v}(\mathbf{p}_k + \mathbf{c}, t + h), \\ \mathbf{p}_{k+1} &= \mathbf{p}_k + (\mathbf{a} + 2\mathbf{b} + 2\mathbf{c} + \mathbf{d})/6, & t &= t + h, \quad k = k + 1. \end{aligned} \quad (5.6)$$

Equation (5.6) is evaluated repeatedly until  $\mathbf{p}_k$  leaves the grid domain  $G$  or  $t \geq t_n$ .

### 5.5.2 Particle Tracing Issues

There are several fundamental issues in particle tracing: physical versus computational space tracing, cell search, step size selection, and velocity interpolation; see Buning [4, 5] and Post and van Walsum [22]. All of these are concerned with the accuracy and speed of particle tracing. In Murman and Powell [20], a first-order integration scheme was shown to produce inaccurate particle traces. They found that using large step size in the integration is undesirable and that the step size should be a fraction of the cell size. Shirayama [27] also concluded that the first-order integration scheme could lead to erroneous results and suggested that a higher-order integration be used. Sadarjoen et al. [25] compared the accuracy and speed of physical space and computational space tracing methods. Trilinear interpolation, inverse distance weighting, and volume weighting schemes for velocity interpolation were evaluated. In Darmofal and Haimes [9], several multistage and multistep integration schemes were compared, and the accuracy and memory requirements of the two schemes were discussed. They concluded that the step size should be adaptive and that a fourth-order interpolation and integration scheme should be used. In the following sections, I review these issues and discuss additional considerations for unsteady flows.

#### Physical Space Versus Computational Space Tracing

In physical space, the grid is curvilinear and the grid cells are warped bricks. The curvilinear grid is defined by a set of points:  $\{(x_{ijk}, y_{ijk}, z_{ijk}), i = 1, \dots, n_x, j = 1, \dots, n_y, k = 1, \dots, n_z\}$ . In flow calculation, the grid is transformed to computational space, where each grid cell is a unit cube. The transformed grid consists of the following set of points:

$\{(i, j, k), i = 1, \dots, n_x, j = 1, \dots, n_y, k = 1, \dots, n_z\}$ . During particle integration, it is necessary to determine the grid cell that a particle currently lies in. This requires cell search (also referred to as point location). In computational space, the grid is uniform and the cell in which the particle currently lies can be easily determined. For example, suppose the computational coordinates of the particle's position are  $(\xi, \eta, \zeta)$ , then the particle lies in grid cell  $(\text{int}(\xi), \text{int}(\eta), \text{int}(\zeta))$ . Although cell search is fast and simple in computational space, there are some disadvantages for tracing in computational space. Firstly, the velocity is usually given in physical space. In order to perform integration in computational space, the velocity needs to be converted into computational space. This requires additional calculation time for the velocity transformation. Secondly, during the transformation, accuracy may be lost due to the transformation scheme used. Lastly, if the grid cell is badly deformed (irregularities exist), then the transformed velocity may be infinite [5]. For these reasons, particle tracing is commonly performed in physical space.

The main reason for tracing in physical space is accuracy. The disadvantage is that cell search is more time consuming in physical space than in computational space. From simple code profiling, Kenwright and Lane found the time spent in cell search could require more than 25 percent of the particle tracing time [15]. Cell search is required whenever the particle moves to a new position. For a multistage integration method such as the RK4 method described earlier, cell search is required at the intermediate stages of the integration. For example, cell search is required for  $\mathbf{p}_k + \mathbf{a}/2$ ,  $\mathbf{p}_k + \mathbf{b}/2$ , and  $\mathbf{p}_k + \mathbf{c}$  in Equations (5.3) and (5.6). If the step size  $h$  is relatively small, then the particle is likely to move within the current cell or jump no more than one cell. Hence, a local cell search can be performed to find the new position. If the grid is multizoned, then a global cell search is required when the particle moves to a new block (referred to as grid jumping). Because global cell search is more computationally expensive than local cell search, grid jumping can increase the particle tracing time considerably. In Hultquist [13], techniques were suggested for speeding up particle tracing in physical space. These include cell caching, extrapolation in cell search, and cell tagging for grid jumping. Recently, Kenwright and Lane [15] were able to improve the speed of particle tracing in physical space by several factors. By decomposing the grid cell into tetrahedra, cell search time was reduced. Furthermore, particle integration, velocity interpolation, and step size control were performed in physical space.

### Cell Search

Given  $\mathbf{p}$  in the physical space, the problem is to find the corresponding point  $\mathbf{c}$  in the computational space. The first step is to find the grid cell that  $\mathbf{p}$  lies in. An intuitive method would be to search for the closest point in the grid using all points. However, this could be expensive if the grid consists of millions of points. Buning [5] suggested searching along edges of the grid cells to find the closest grid point, and then using a "stencil walk" approach to find the exact offset of the particle inside the grid cell. The stencil walk approach, which is based on the Newton-Raphson approach, is summarized below:

1. Select the center of the grid cell  $(i, j, k)$  as the initial guess of  $\mathbf{c}$ , where  $(i, j, k)$  is the closest grid point to  $\mathbf{p}$ . Thus, let  $\mathbf{c} = (\xi, \eta, \zeta)$ , where  $\xi = i + 0.5$ ,  $\eta = j + 0.5$ , and  $\zeta = k + 0.5$ .

- Convert  $(\xi, \eta, \zeta)$  to its corresponding physical point  $\mathbf{p}(\xi, \eta, \zeta)$  using trilinear interpolation.

$$\begin{aligned} \mathbf{p}(\xi, \eta, \zeta) = & [(\mathbf{p}_{i,j,k}(1-\alpha) + \mathbf{p}_{i+1,j,k}\alpha)(1-\beta) \\ & + (\mathbf{p}_{i,j+1,k}(1-\alpha) + \mathbf{p}_{i+1,j+1,k}\alpha)\beta](1-\gamma) \\ & + [(\mathbf{p}_{i,j,k+1}(1-\alpha) + \mathbf{p}_{i+1,j,k+1}\alpha)(1-\beta) \\ & + (\mathbf{p}_{i,j+1,k+1}(1-\alpha) + \mathbf{p}_{i+1,j+1,k+1}\alpha)\beta]\gamma, \end{aligned} \quad (5.7)$$

where  $\alpha = \xi - i$ ,  $\beta = \eta - j$ , and  $\gamma = \zeta - k$ .

- Compute the difference vector  $\Delta \mathbf{p}$ , where  $\Delta \mathbf{p} = \mathbf{p} - \mathbf{p}(\xi, \eta, \zeta)$ . This vector indicates how close  $\mathbf{p}(\xi, \eta, \zeta)$  is to  $\mathbf{p}$  in physical space.
- Convert  $\Delta \mathbf{p}$  to  $\Delta \mathbf{c}$ , where  $\Delta \mathbf{c}$  is the difference vector mapped into computational space. Let  $\Delta \mathbf{p} = (\Delta x, \Delta y, \Delta z)$  and  $\Delta \mathbf{c} = (\Delta \alpha, \Delta \beta, \Delta \gamma)$ . Then,

$$\begin{aligned} \begin{bmatrix} \Delta \alpha \\ \Delta \beta \\ \Delta \gamma \end{bmatrix} &= J^{-1} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix}, \quad \text{where} \\ J &= \begin{bmatrix} x_\xi & x_\eta & x_\zeta \\ y_\xi & y_\eta & y_\zeta \\ z_\xi & z_\eta & z_\zeta \end{bmatrix} \quad \text{and} \quad J^{-1} = \begin{bmatrix} \xi_x & \xi_y & \xi_z \\ \eta_x & \eta_y & \eta_z \\ \zeta_x & \zeta_y & \zeta_z \end{bmatrix}. \end{aligned} \quad (5.8)$$

The metric terms in  $J^{-1}$  are

$$\begin{aligned} \xi_x &= D(y_\eta z_\zeta - y_\zeta z_\eta), \quad \xi_y = D(x_\zeta z_\eta - x_\eta z_\zeta), \quad \xi_z = D(x_\eta y_\zeta - x_\zeta y_\eta), \\ \eta_x &= D(y_\zeta z_\xi - y_\xi z_\zeta), \quad \eta_y = D(x_\xi z_\zeta - x_\zeta z_\xi), \quad \eta_z = D(x_\zeta y_\xi - x_\xi y_\zeta), \\ \zeta_x &= D(y_\xi z_\eta - y_\eta z_\xi), \quad \zeta_y = D(x_\eta z_\xi - x_\xi z_\eta), \quad \zeta_z = D(x_\xi y_\eta - x_\eta y_\xi), \end{aligned} \quad (5.9)$$

where  $D$  is the determinant of the Jacobian matrix  $J$  and

$$D = 1/(x_\xi y_\eta z_\zeta - x_\xi y_\zeta z_\eta - y_\xi x_\eta z_\zeta + x_\zeta y_\xi z_\eta + x_\eta y_\zeta z_\xi - x_\zeta y_\eta z_\xi). \quad (5.10)$$

The partial derivatives in the Jacobian matrix  $J$  are the partial derivatives of (5.7), where  $\mathbf{p} = (p_x, p_y, p_z)$ . For example,  $x_\xi = \partial p_x / \partial \xi$ ,  $y_\xi = \partial p_y / \partial \xi$ ,  $z_\xi = \partial p_z / \partial \xi$ , and so on.

- Let  $\alpha = \alpha + \Delta \alpha$ ,  $\beta = \beta + \Delta \beta$ , and  $\gamma = \gamma + \Delta \gamma$ . If  $\alpha, \beta, \gamma \notin [0,1]$ , then  $\mathbf{p}$  is outside the current cell. Increase  $i$  by 1 if  $\alpha > 1$  or decrease  $i$  by 1 if  $\alpha < 0$ . Update  $j$  and  $k$  according to  $\beta$  and  $\gamma$ , respectively, then go to step 1.
- Let  $\xi = \xi + \Delta \alpha$ ,  $\eta = \eta + \Delta \beta$ ,  $\zeta = \zeta + \Delta \gamma$ . If  $|\Delta \mathbf{c}| < \epsilon$ , where  $\epsilon$  is the chosen tolerance, then  $\mathbf{p}(\xi, \eta, \zeta)$  is close enough to  $\mathbf{p}$  and its corresponding point  $(\xi, \eta, \zeta)$  in computational space has been found. Otherwise, go to step 2.

### Step Size Selection

The distance that the particle traverses at each integration step is based on the step size  $h$  and the velocity at  $\mathbf{p}$ . The larger  $h$  is, the further  $\mathbf{p}$  traverses. If  $h$  is too large, then the resulting particle trace can be inaccurate because the particle may have missed important flow features. This is especially true if the flow changes direction rapidly. Likewise, if  $h$  is too small, then particles may unnecessarily take too many steps to traverse the grid, which would increase the computation time. A good rule for selecting  $h$  is based on the velocity at the current grid cell: If the velocity is large, then  $h$  should be small. Buning [5] suggested letting  $h = c/\max(|U|, |V|, |W|)$ , where  $(U, V, W)$  represent the computational velocity at  $\mathbf{p}$  and

$$\begin{bmatrix} U \\ V \\ W \end{bmatrix} = J^{-1} \begin{bmatrix} u \\ v \\ w \end{bmatrix}. \quad (5.11)$$

Matrix  $J^{-1}$  is given in (5.8). The computational velocity is used so that the number of steps in each cell is consistent. For example, if  $c = 0.2$ , then the particle will traverse no more than one-fifth of a computational cell at each step. Small  $c$  yields small steps. A common scheme for adaptively setting  $h$  is step doubling, which successively reduces  $h$  until some desired accuracy is obtained (see Press et al. [10]). Another scheme for determining  $h$  is to base it on the curvature of the particle trace. If the curvature is high, then  $h$  should be small.

### Velocity Interpolation

In particle tracing, the velocity at the current position of the particle is required to advance the particle. Velocity interpolation is performed at each stage of the RK4 integration. The velocity at  $\mathbf{p}$  can be interpolated using the velocities at the corners of the grid cell that contains  $\mathbf{p}$ . A fast and simple scheme is trilinear interpolation. If  $\mathbf{p}$  is in grid cell  $(i, j, k)$  and  $\mathbf{p}$  has the fractional offsets  $(\alpha, \beta, \gamma)$  from the grid point at  $(i, j, k)$ , then

$$\begin{aligned} \mathbf{v}(i + \alpha, j + \beta, k + \gamma) = & [(\mathbf{v}_{i,j,k}(1 - \alpha) + \mathbf{v}_{i+1,j,k}\alpha)(1 - \beta) \\ & + (\mathbf{v}_{i,j+1,k}(1 - \alpha) + \mathbf{v}_{i+1,j+1,k}\alpha)\beta](1 - \gamma) \\ & + [(\mathbf{v}_{i,j,k+1}(1 - \alpha) + \mathbf{v}_{i+1,j,k+1}\alpha)(1 - \beta) \\ & + (\mathbf{v}_{i,j+1,k+1}(1 - \alpha) + \mathbf{v}_{i+1,j+1,k+1}\alpha)\beta]\gamma, \end{aligned} \quad (5.12)$$

where  $\mathbf{v}_{i,j,k}$  is the velocity at grid point  $(i, j, k)$ . The trilinear interpolant assumes that the velocity varies linearly across the edges of the cell. Though trilinear interpolation is simple, accuracy may be lost if the grid cell is deformed. In Yeung and Pope [32], higher-order interpolations were compared to linear interpolation. They showed that cubic spline interpolation gives more accurate results than linear interpolation in turbulent flows, and that inaccuracy due to interpolation was greater than that due to integration when small step size is used. The disadvantage of using a cubic spline interpolant is that it is more expensive and requires velocities from 64 nodes versus eight nodes.

Steady flows do not change in time and we can assume that the number of time steps is infinite. Thus, the velocity function is defined for any  $t$ . However, unsteady flows vary in

time and the velocity function is known only at time steps  $t_1, \dots, t_n$ . At time  $t$ , if  $t \neq t_l$  for  $l = 1, \dots, n$ , then a temporal interpolation of velocity is performed prior to the spatial interpolation given in (5.12). If  $t_l \leq t \leq t_{l+1}$ , then let

$$\mathbf{v}_{i,j,k}(\delta) = (1 - \delta)\mathbf{v}_{i,j,k}^{t_l} + \delta\mathbf{v}_{i,j,k}^{t_{l+1}} \quad \text{and} \quad \delta = (t - t_l)/(t_{l+1} - t_l),$$

where  $\mathbf{v}_{i,j,k}^{t_l}$  and  $\mathbf{v}_{i,j,k}^{t_{l+1}}$  are the velocities at grid point  $(i, j, k)$  at time  $t_l$  and  $t_{l+1}$ , respectively. After the above temporal interpolation of velocity, Equation (5.12) can then be evaluated by letting  $\mathbf{v}_{i,j,k} = \mathbf{v}_{i,j,k}(\delta)$ . The above interpolation assumes that the flow varies linearly between the time steps and that it is only second-order accurate in time over a complete RK4 integration. Darmofal and Haines [9] suggested a double time-step RK4 method, which is fourth-order accurate in time and does not require temporal interpolation of velocity. The method uses the velocities at the given time steps for the intermediate stages of the RK4 integration, and  $h$  is the delta time between two consecutive time steps. Thus, Equation (5.6) becomes

$$\begin{aligned} \mathbf{a} &= 2h\mathbf{v}(\mathbf{p}_k, t), & \mathbf{b} &= 2h\mathbf{v}(\mathbf{p}_k + \mathbf{a}/2, t + h), \\ \mathbf{c} &= 2h\mathbf{v}(\mathbf{p}_k + \mathbf{b}/2, t + h), & \mathbf{d} &= 2h\mathbf{v}(\mathbf{p}_k + \mathbf{c}, t + 2h), \\ \mathbf{p}_{k+1} &= \mathbf{p}_k + (\mathbf{a} + 2\mathbf{b} + 2\mathbf{c} + \mathbf{d})/6, & t &= t + 2h, \quad k = k + 1. \end{aligned}$$

Although this method is attractive, the step size is uniform and the accuracy of the integration is dictated by the temporal resolution of the saved time steps. For large unsteady flow data sets, the data are usually saved at some fixed time interval (such as every 50th time step) which may not have sufficient resolution for this method. Furthermore, particles are saved only at every other time step.

### Particle Tracing in Multizoned Grids

When tracing particles in a multizoned grid, special consideration is required to track the particles from one block to another. A common grid generation technique used for multizoned grids is the 3D Chimera grid-embedding scheme [2]. The output grid generated by this scheme is used by PLOT3D and FAST. The Chimera scheme stores an integer code called *iblack* at each grid point. The *iblack* of a grid point indicates whether the grid point lies in more than one block. The *iblack* may also indicate whether the grid point is near a wall boundary or that the velocity at a grid point is invalid. When a particle leaves the current block, the *iblack*s at the corners of the grid cell are checked to determine if the particle can continue to another block. If this is possible, then a global cell search is performed to find the grid cell in the new block where the particle lies.

### Particle Tracing in Moving Grids

For unsteady flow simulations, the grid may move in time. Particle tracing in moving grids requires additional interpolations. In cell search, to find the current grid cell containing  $\mathbf{p}$  at time  $t$ , an interpolated grid cell is generated. The grid cell is simply a linear interpolation of the grid cells at  $t_l$  and  $t_{l+1}$  if  $t_l \leq t \leq t_{l+1}$ . It is not necessary to interpolate the entire grid, only the current grid cell in which  $\mathbf{p}$  lies. Thus, at each intermediate stage of the RK4 integration, an interpolated grid cell is computed for velocity interpolation and cell search.

To transform the velocity from physical space to computational space in unsteady flows with moving grids, Equations (5.8) and (5.11) need to be modified to consider  $t$  [24] and the grid velocity  $(x_\tau, y_\tau, z_\tau)$ . Let

$$\begin{bmatrix} U \\ V \\ W \\ T \end{bmatrix} = J^{-1} \begin{bmatrix} u \\ v \\ w \\ t \end{bmatrix},$$

$$J = \begin{bmatrix} x_\xi & x_\eta & x_\zeta & x_\tau \\ y_\xi & y_\eta & y_\zeta & y_\tau \\ z_\xi & z_\eta & z_\zeta & z_\tau \\ t_\xi & t_\eta & t_\zeta & t_\tau \end{bmatrix} \quad \text{and} \quad J^{-1} = \begin{bmatrix} \xi_x & \xi_y & \xi_z & \xi_t \\ \eta_x & \eta_y & \eta_z & \eta_t \\ \zeta_x & \zeta_y & \zeta_z & \zeta_t \\ \tau_x & \tau_y & \tau_z & \tau_t \end{bmatrix}.$$

The partial derivatives  $t_\xi, t_\eta, t_\zeta$  are all zero,  $t_\tau = t_{l+1} - t_l$ , and

$$(x_\tau, y_\tau, z_\tau) = \mathbf{p}^{t_{l+1}}(\xi, \eta, \zeta) - \mathbf{p}^{t_l}(\xi, \eta, \zeta),$$

where  $\mathbf{p}^{t_l}(\xi, \eta, \zeta)$  and  $\mathbf{p}^{t_{l+1}}(\xi, \eta, \zeta)$  are  $\mathbf{p}(\xi, \eta, \zeta)$  at time  $t_l$  and  $t_{l+1}$  respectively. The nine metric terms in the upper-left corner of  $J^{-1}$  are the same as those given in (5.9), and the remaining terms are

$$\begin{aligned} \xi_t &= (-x_\tau \xi_x - y_\tau \xi_y - z_\tau \xi_z)/t_\tau D, & \eta_t &= (-x_\tau \eta_x - y_\tau \eta_y - z_\tau \eta_z)/t_\tau D, \\ \zeta_t &= (-x_\tau \zeta_x - y_\tau \zeta_y - z_\tau \zeta_z)/t_\tau D, & \tau_x &= 0, \quad \tau_y = 0, \quad \tau_z = 0, \quad \text{and} \quad \tau_t = 1/t_\tau, \end{aligned}$$

where the determinant  $D$  is given in (5.10).

## 5.6 Unsteady Flow Analysis Toolkit

A particle tracing system called Unsteady Flow Analysis Toolkit (UFAT) has been developed to compute particle traces in unsteady flows [18]. UFAT has been used to generate streaklines from several large 3D unsteady flows with multizoned curvilinear grids. Some of these grids move in time. UFAT was developed specifically for unsteady flows with a large number of time steps. The major features in UFAT are listed below:

- Computes streamlines, streaklines, pathlines, and timelines
- Performs particle tracing in multizoned curvilinear grids with rigid-body motion
- Allows particle tracing of unsteady flows with hundreds of time steps
- Assigns color to particles based on position, time, or a scalar quantity
- Provides RK2 and RK4 integration schemes with adaptive step size
- Allows particle tracing restricted to a grid surface (oil flow)
- Saves particle traces to a graphics metafile for playback
- Provides save-and-restore option for nonconsecutive run sessions. This feature allows particle traces to be computed from many time steps without requiring all time steps of the flow data to be on line at the same time.

### 5.6.1 Implementation

Because it is usually impossible to keep all time steps of an unsteady flow data set in memory, UFAT stores only two consecutive time steps of the data in memory. The flow data at time steps  $t_l$  and  $t_{l+1}$  are used to integrate particles from  $t_l$  to  $t_{l+1}$ . UFAT uses PLOT3D's particle tracing library for cell search and grid jumping algorithms. I have modified the library to support these and other algorithms in unsteady flows.

UFAT releases particles from the user-specified seed locations at a given time-step interval and tracks the particles. For streaklines, particles are released from the seed locations at each time step. For streamlines, particle traces are computed for the specified seed locations at the given time steps; thus, a set of streamlines is computed for each time step. For pathlines, particles are released from the seed locations at the first time step only, and are then tracked through the given time steps. Pathlines show the trajectories of particles released from the seed locations. For timelines, particles are tracked in the same manner as streaklines, however, they are represented in a different form. For timelines, particles that are released simultaneously are connected (see Figure 5.3).

UFAT currently runs on the Cray, Convex, and SGI systems. The output of UFAT is a graphics metafile. This metafile can then be rendered using a graphics program. Currently, the metafile is written in a format that can be displayed by FAST.

### 5.6.2 Algorithms

Below is a basic procedure for tracing a given particle  $\mathbf{p}$  in the grid domain  $G$  and the steady velocity field  $\mathbf{v}$ .

```

Procedure Trace_Particle( p, G, v )
  c = Search_Cell( p, G )
  While p ∈ G do
    v_p = Interpolate( p, c, G, v )
    h = Compute_Step_Size( p, c, G, v_p )
    p = Advance( p, h, v_p, G )
    c = Search_Cell( p, G )
  End While

```

`Search_Cell()` searches for the grid cell that  $\mathbf{p}$  lies in. `Interpolate()` interpolates the velocity at  $\mathbf{p}$ . Based on the interpolated velocity at  $\mathbf{p}$ , `Compute_Step_Size()` computes the step size  $h$ . `Advance()` advances particle  $\mathbf{p}$  using the RK4 method described in Section 5.5.1. For time-dependent particle tracing in a static grid, the above procedure needs to be modified to consider velocity as a function of time. Let  $\Omega = \{(t_l, \mathbf{v}_l), l = 1, \dots, n\}$ , where  $\mathbf{v}_l$  is the velocity field at time  $t_l$  and  $n$  is the number of time steps in the unsteady flow. The procedure for tracing particle  $\mathbf{p}$  from time  $t_1$  to  $t_n$  is given below.

```

Procedure Trace_Time_Dependent_Particle(p, G, Ω)
  t = t1
  l = 1
  c = Search_Cell( p, G )
  While (p ∈ G) and (t < tn) do
    v_p = Interpolate(p, c, G, t, tl, tl+1, vl, vl+1 )

```

```

h = Compute_Step_Size( p, c, G, v_p )
p = Advance( p, h, v_p, G, t, t_l, t_{l+1}, v_l, v_{l+1} )
c = Search_Cell( p, G )
t = t + h
if ( t > t_{l+1} ) l = l + 1
End While

```

In `Interpolate()`, the velocity is interpolated in time and space. If the grid moves in time (unsteady grid), then it is also necessary to interpolate the grid cell at each  $t$ . This implies that `Search_Cell()` also needs to interpolate the grid cell in time.

### 5.6.3 Summary of Steps for Streaklines

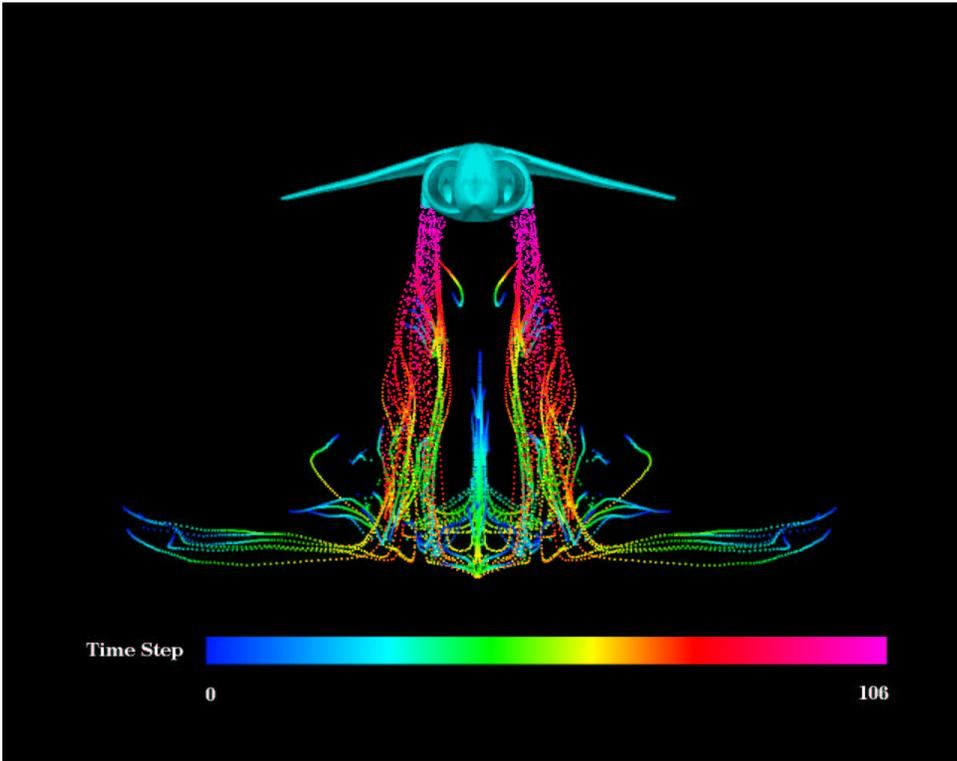
Below is a summary of the steps for computing streaklines from an unsteady flow data set.

1. Release one particle from every seed location.
2. Read the first time step's grid and solution files.
3. For the remaining time steps, do the following:
  - 3.1 Read the current time step's grid (if unsteady grid) and solution files.
  - 3.2 Advance all particles from the previous time step to the current time step.
  - 3.3 Release one particle from every seed location.
  - 3.4 Compute color values of all particles based on their positions, time at release, or a specified scalar quantity.
  - 3.5 Save all active particles and their color values to the graphics metafile.

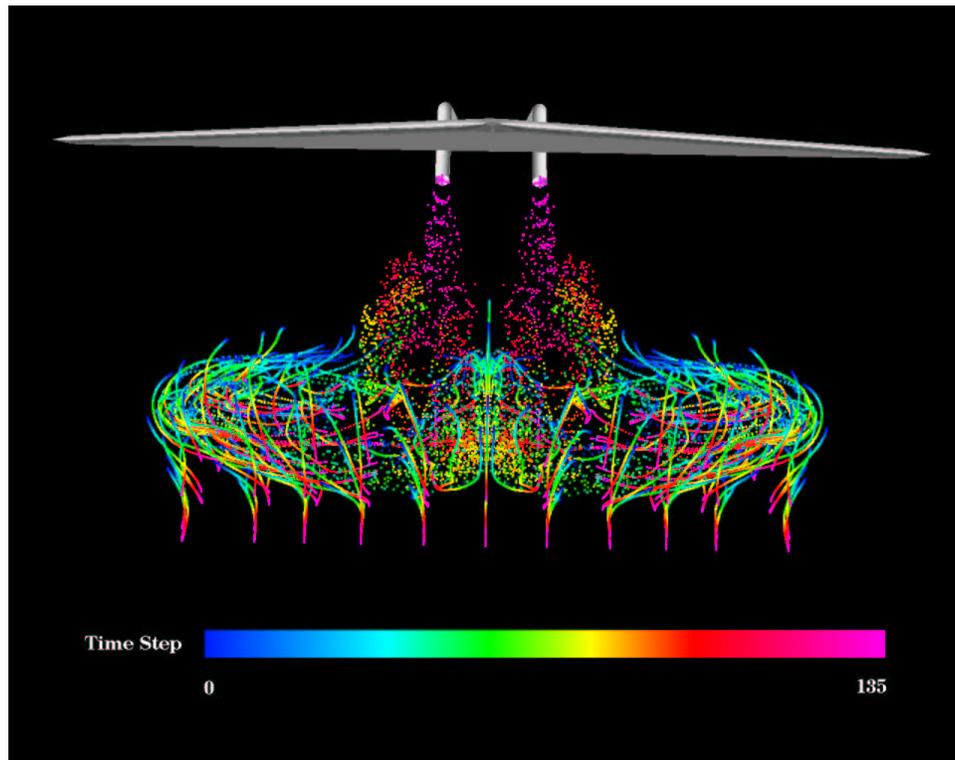
## 5.7 Examples

This section depicts streaklines computed from several unsteady flow data sets given in Section 5.2. The flow calculations were performed on Cray YMP and C90 supercomputers. The streaklines were computed using UFAT on a Convex C3240 system and animated using FAST on a Silicon Graphics Reality Engine. The streaklines shown in this section are snapshots from the animation, which is an effective way to visualize streaklines. The streaklines are represented by individual particles instead of "connected" particles, because when adjacent particles are too far apart, streaklines can become jagged. Figure 5.4 shows streaklines computed from a simulation of the Harrier jet at time step 106. Particles, which are released from the two exhaust pipes of the jet, are colored by time at release. Blue represents the earliest time and magenta represents the most recent time.

The second example is a delta wing in descent. For this simulation, the effects of two thrust-reverser jets in slow-speed flight near the ground are studied (Chawla and van Dalsem [8]). Figure 5.5 shows streaklines surrounding the delta wing at time step 135. The particles are colored by time at release. At each time step, particles are released near the ground and the two jet exits. In the animation, the interactions of the particles released from the jet exits and those released near the ground are clearly visible.

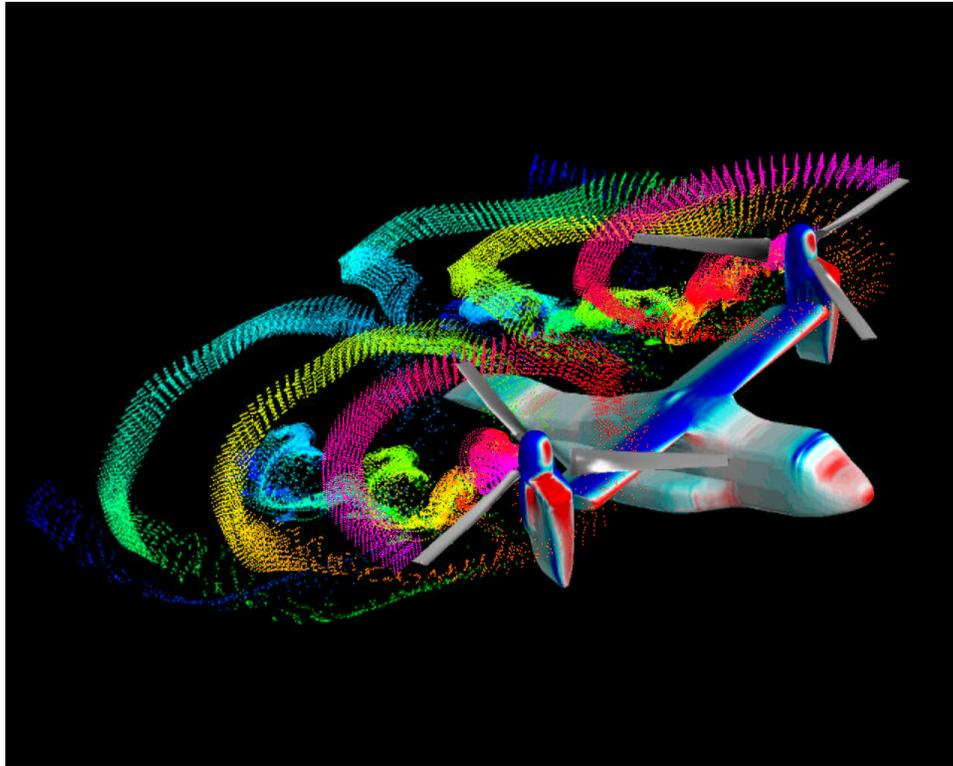


**Figure 5.4:** Streaklines released from two jet exits of the Harrier jet. Particles are colored by time at release.



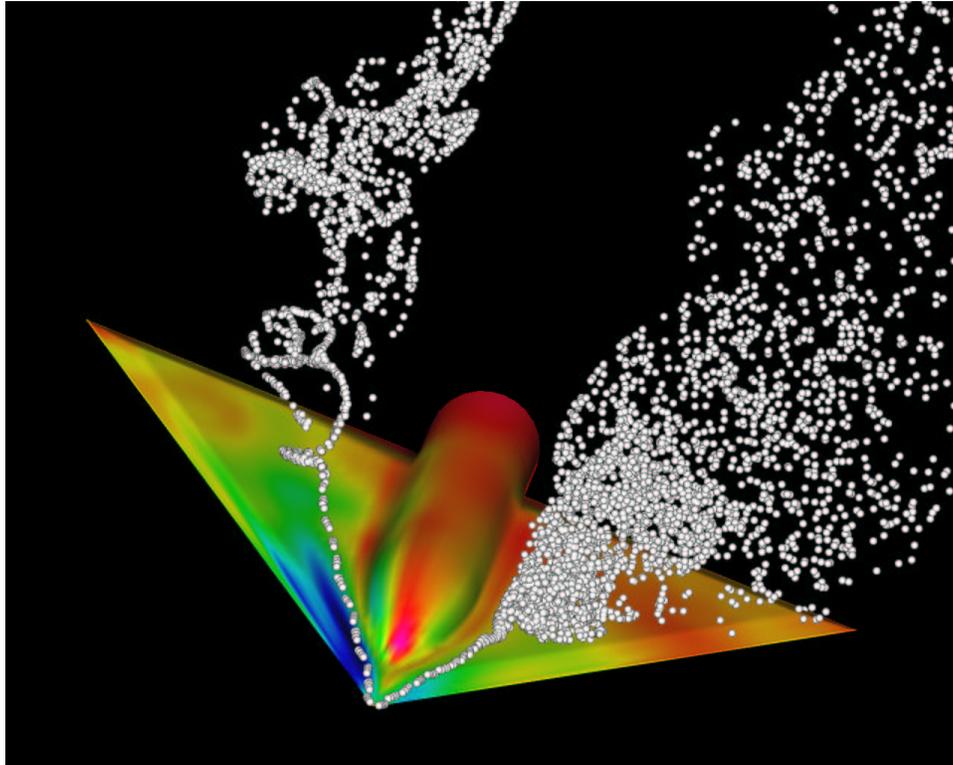
**Figure 5.5:** Interactions of particles released from the two jets of a descending delta wing and those released near the surface of the ground are shown. Particles are colored by time at release.

Figure 5.6 shows streaklines surrounding the V-22 tilt rotor aircraft after three blade revolutions. The V-22 aircraft has two propellers that rotate in opposite directions. The simulation studies the use of Chimera overset grid methods for computing viscous flow about a complete tilt rotor aircraft (Meakin [19]). In the simulation, there are 1,450 time steps in one blade revolution and each time step requires 52 MB for the grid and solution files (see Table 5.2). Thus, a total of 75.4 GB is required per revolution. Three revolutions are calculated in the simulation. Because there was not enough disk space to store 226.2 GB ( $3 \times 75.4$  GB) of flow data, the data were saved at every 15th time step during the flow calculation. This results in 97 time steps per revolution and 15 GB ( $3 \times 97 \times 52$  MB) of data were used for the visualization. Particles are released near a rotor blade and the nacelle. In Figure 5.6, the surface of the V-22 aircraft is colored by pressure and the particles are colored by time at release. At each time step, 400 particles are released.



**Figure 5.6:** Streaklines surrounding the V-22 tilt rotor aircraft after three blade revolutions. The V-22 is colored by pressure and the particles are colored by time at release.

Figure 5.7 shows particles released near the noise of a delta wing with wing rock motion. Particles are represented by spheres to give a better depth perception. The surface of the delta wing is colored by pressure. The simulation is for the development of an experimentally validated CFD tool, which will be used to predict and analyze high-angle-of-attack maneuver aerodynamics (Chaderjian and Schiff [7]).



**Figure 5.7:** Particles released near the noise of a delta wing with wing rock motion. Spiral flow is evident above the left wing and vortex breakdown is visible above the right wing.

## 5.8 Conclusions

Unsteady flow visualization is a relatively new problem in scientific visualization. The time-dependent nature and the large-scale magnitude of the data make unsteady flow visualization challenging and interesting. Because most existing flow visualization techniques were developed for steady flow data, there is a current need for unsteady flow visualization techniques. The technique described in this chapter uses streaklines to depict time-varying phenomena in unsteady flows. Presently, unsteady flow visualization is likely to rely on scripting and subsampling approaches because of limited hardware capabilities. The size of unsteady flow data sets will continue to increase in the future. There is a continuing need to increase the storage, networking, and computing capabilities as numerical flow simulations become more complex.

## Acknowledgments

This work was performed in the Numerical Aerodynamic Simulation Systems Division at NASA Ames Research Center under contract NAS 2-12961. I thank Jill Dunbar, David

Kenwright, Gregory Nielson, and the reviewers for their helpful comments. I also thank many colleagues at the NAS division for their support. I thank the following CFD scientists for providing their data sets: Neal Chaderjian and Lewis Schiff (rolling delta wing), Kalpana Chawla (descending delta wing), Sungho Ko (oscillating airfoil), Robert Meakin (V-22 aircraft), and Merritt Smith (Harrier jet). The geometry of the V-22 aircraft was provided by Bell Helicopter Textron Inc. and Boeing Helicopters.

## Bibliography

- [1] G. Bancroft, F. Merritt, T. Plessel, P. Kelaita, K. McCabe, and A. Globus. FAST: A multi-processed environment for visualization of computational fluid dynamics. In A. Kaufman, editor, *Proceedings of Visualization '90*, pages 14–27, San Francisco, CA, Oct. 1990.
- [2] J. Benek, P. Buning, and J. Steger. A 3-d chimera grid embedding technique. In *7th Computational Fluid Dynamics Conference*, number AIAA 85-1523, Cincinnati, OH, July 1985.
- [3] S. Bryson and C. Levit. The virtual wind tunnel. *IEEE Computer Graphics and Applications*, 12(4):25–34, July 1992.
- [4] P. Buning. Sources of error in the graphical analysis of CFD results. *Journal of Scientific Computing*, 3(2):149–164, 1988.
- [5] P. Buning. Numerical algorithms in CFD post-processing, computer graphics and flow visualization in computational fluid dynamics, von karman institute for fluid dynamics lecture series 1989-07, 1989.
- [6] P. Buning and J. Steger. Graphics and flow visualization in computational fluid dynamics. In *7th Computational Fluid Dynamics Conference*, number AIAA 85-1507, Cincinnati, Ohio, July 1985.
- [7] N. Chaderjian and L. Schiff. Navier-stokes prediction of large-amplitude forced and free-to-roll delta-wing oscillations, AIAA paper 94-1884-cp, June 1994.
- [8] K. Chawla and W. van Dalsem. Numerical simulation of stol operations using thrust-vectoring. In *AIAA Aircraft Design Systems Meeting*, number AIAA 92-4254, Hilton Head, SC, Aug. 1992.
- [9] D. Darmofal and R. Haimes. An analysis of 3-D particle path integration algorithms. Submitted to *12th Computational Fluid Dynamics Conference*, San Diego, CA, June 1995.
- [10] W. Press et al. *Numerical Recipes*. Cambridge University Press, 1986.
- [11] R. Haimes. pV3: A distributed system for large-scale unsteady CFD visualization. Technical report, 32nd AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada, Jan. 1994.

- [12] R. Haimes and M. Giles. VISUAL3: Interactive unsteady unstructured 3d visualization. Technical Report AIAA 91-0794, 29th AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada, Jan. 1991.
- [13] J. Hultquist. Improving the performance of particle tracing in curvilinear grids. Technical Report AIAA 94-0324, 32nd AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada, Jan. 1994.
- [14] D. Jespersen and C. Levit. Numerical simulation of flow past a tapered cylinder. Technical Report AIAA 91-0801, 29th AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada, Jan. 1991.
- [15] D. Kenwright and D. Lane. Optimization of time-dependent particle tracing using tetrahedral decomposition. In G. Nielson and D. Silver, editors, *Proceedings of Visualization '95*, pages 321–328, Atlanta, GA, Oct. 1995.
- [16] D. Kenwright and G. Mallinson. A 3-D streamline tracking algorithm using dual stream functions. In A. Kaufman and G. Nielson, editors, *Proceedings of Visualization '92*, pages 62–68, Boston, MA, Oct. 1992.
- [17] D. Lane. Visualization of time-dependent flow fields. In G. Nielson and D. Bergeron, editors, *Proceedings of Visualization '93*, pages 32–38, San Jose, CA, Oct. 1993.
- [18] D. Lane. UFAT—a particle tracer for time-dependent flow fields. In D. Bergeron and A. Kaufman, editors, *Proceedings of Visualization '94*, pages 257–264, Washington, DC, Oct. 1994.
- [19] R. Meakin. Moving body overset grid methods for complete aircraft tiltrotor simulations. In *11th AIAA Computational Fluid Dynamics Conference*, Orlando, FL, July 1993.
- [20] E. Murman and K. Powell. Trajectory integration in vortical flows. *AIAA Journal*, 27(7):982–984, July 1989.
- [21] H. Pagendarm. HIGHEND, a visualization system for 3d data with special support for postprocessing of fluid dynamics data. In M. Grave, Y. LeLous, and W. Hewitt, editors, *Visualization in Scientific Computing*. Springer-Verlag, Heidelberg, 1993.
- [22] F. Post and T. van Walsum. Fluid flow visualization. In H. Hagen, H. Mueller, and G. Nielson, editors, *Focus on Scientific Visualization*, pages 1–40. Springer, Berlin, 1993.
- [23] F. Post and J. van Wijk. Visual representation of vector fields: Recent developments and research directions. In L. Rosenblum et al., editor, *Scientific Visualization: Advances and Challenges*, pages 367–390. Academic Press, San Diego, 1993.
- [24] T. Pulliam. Efficient solution methods for navier-stokes equations, von karman institute for fluid dynamics lecture series, 1986.

- 
- [25] A. Sadarjoen, T. van Walsum, A. Hin, and F. Post. Particle tracing algorithms for 3D curvilinear grids. In *5th Eurographics Workshop on Visualization in Scientific Computing*, Rostock, Germany, May 1994.
- [26] H. Schlichting. *Boundary Layer-Theory*. McGraw Hill, New York, 1979.
- [27] S. Shirayama. Processing of computed vector fields for visualization. *Journal of Computational Physics*, 106:30–41, 1993.
- [28] M. Smith, W. van Dalsem, F. Dougherty, and P. Buning. Analysis and visualization of complex unsteady three-dimensional flows. Technical Report AIAA 89-0139, 27th AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada, Jan. 1989.
- [29] D. Speray and S. Kennon. Volume probes. *Proceedings of San Diego Workshop on Volume Visualization, Computer Graphics*, 24(5):5–12, Nov. 1990.
- [30] A. Vaziri, M. Kremenetsky, M. Fitzgibbon, and C. Levit. Experiences with CM/AVS to visualize and compute simulation data on the CM-5. NAS Applied Research Technical Report RNR 94-005, NASA Ames Research Center, 1994.
- [31] H. Vollmers. The recovering of flow features from large numerical data bases, vki lecture series on computer graphics and flow visualization in computational fluid dynamics, von karman institute for fluid dynamics lecture series, 1991.
- [32] P. Yeung and S. Pope. An algorithm for tracking fluid particles in numerical simulations of homogeneous turbulence. *Journal of Computational Physics*, 79:373–416, 1988.