UFAT — A Particle Tracer for Time-Dependent Flow Fields

David A. Lane

Computer Sciences Corporation
NASA Ames Research Center
M/S T27A-2
Moffett Field, CA 94035

Abstract

Time-dependent (unsteady) flow fields are commonly generated in Computational Fluid Dynamics (CFD) simulations; however, there are very few flow visualization systems that generate particle traces in unsteady flow fields. Most existing systems generate particle traces in time-independent flow fields. A particle tracing system has been developed to generate particle traces in unsteady flow fields. The system was used to visualize several 3D unsteady flow fields from real-world problems, and it has provided useful insights into the time-varying phenomena in the flow fields. In this paper, the design requirements and the architecture of the system are described. Some examples of particle traces computed by the system are also shown.

1 Introduction

Particle systems were introduced in [14] to model fuzzy objects like fire, clouds, and water. For this type of particle system, the motion of the particle is based on some stochastic model. Extensions to this type of particle system have included modeling of snow, grass, smoke, and fireworks. In CFD, particle traces can be used to visualize several time-varying phenomena in the flow field. For example, vortex shedding, formation, and separation [15]. When particle traces are used in this context, the motion of the particle is based on the physical velocity from the flow field.

An *instantaneous streamline* is a curve that is tangent to the vector field at an instant in time. In time-independent (steady) flow, instantaneous streamlines are computed from the flow field at an instant in time. A *streakline* is a line joining the positions at an instant in time of all particles that have been released from a fixed location, called the *seed location*. In unsteady flow, streaklines are computed from several thousand time steps. Streaklines are commonly simulated by releasing particles continuously from the seed locations at each time step. In hydrodynamics, streaklines are simulated by releasing hydrogen bubbles rapidly from the seed locations. Instantaneous streamlines and streaklines are identical in steady flow fields.

In this paper, I introduce a particle tracing system called Unsteady Flow Analysis Toolkit (UFAT), which generates particle traces in unsteady flow fields. UFAT differs from existing systems in that it computes streaklines from a large number of time steps, performs particle tracing in flow fields with moving grids, provides a save/restore option, and supports playback. Preliminary results of UFAT were presented in [10]. This paper describes the design requirements and the architecture of UFAT. First, the basic problem of particle tracing in unsteady flow fields is described. The design requirements and the particle tracing algorithms of UFAT are then described. Examples of streaklines computed for three real-world problems are shown, and the performance of UFAT is analyzed. Finally, future enhancements for UFAT are discussed.

2 Particle Tracing

The basic problem of particle tracing can be stated as follows: assume that a vector function $\vec{V}(p,t)$ is defined for all p in the domain D and $t \in [t_1, t_n]$, where n is the number of time steps in the unsteady flow. For any particle $p \in D$, find the path of p. The path of p is governed by the following equation:

$$\frac{dp}{dt} = \vec{V}(p,t). \tag{1}$$

The path of p can be found by numerically integrating Equation (1). Several schemes can be used to integrate the above equation. A common scheme is the

1070-2385/94 $04.00 © 1994 IEEE

second-order Runge-Kutta integration with adaptive stepsizing. Let p_0 be the initial point (the seed location) of the particle p and $k = 0$. Then,

$$p^* = p_k + h\vec{V}(p_k, t),$$

$$p_{k+1} = p_k + h(\vec{V}(p_k, t) + \vec{V}(p^*, t + h))/2,$$

$$t = t + h \qquad \text{and} \qquad k = k + 1, \tag{2}$$

where $h = c/max(\vec{V}(p_k))$, $max()$ is the maximum velocity component of $\vec{V}(p_k)$, and $0 < c \leq 1$. The constant c controls the step size of the particle. If c is small, then the particle will traverse many steps in the grid cell. Small values of c should be used for grid regions with rapidly varying velocity. Otherwise, the particle p may advance out of the domain in just a few steps. The integration scheme stated above can be performed in the physical coordinate space or in the computational coordinate space. If the integration is performed in computational space, then it is simple and fast. During the integration, a cell search operation is performed to determine the grid cell that p_{k+1} lies in. Since the grid domain D is rectilinear in computational space, the grid cell can be easily determined by taking the integer computational coordinates of p_{k+1}. For example, if the computational coordinates of p_{k+1} are (ξ, η, ζ), then p_{k+1} lies in grid cell ($\mathtt{int}(\xi)$, $\mathtt{int}(\eta)$, $\mathtt{int}(\zeta)$). In the physical coordinate space, the grid domain D is curvilinear. The cell search operation usually requires performing an iterative algorithm to find the cell that p_{k+1} lies in. A common algorithm used is the Newton-Raphson method. Although particle integration can be done faster in the computational coordinate space than in the physical coordinate space, integrating in computational space may be inaccurate if there are singularities in the grid. For computational space integration, physical velocities are transformed into computational velocities. Singularities in the grid could result in infinite transformed velocities [4]. For this reason, UFAT performs particle integration in physical space.

If the grid is moving in time, then p_{k+1} is likely to be in a grid cell different from the cell that p_k lies in. To determine the cell that p_{k+1} lies in, the cell search operation discussed above is performed. If the grid consists of several blocks (a type of grid known as a *multi-block grid*), then p_{k+1} may lie in a block different from the block that p_k lies in. If p_k is near the boundary of a block, then it is necessary to check if p_{k+1} will be in a different block. This also requires a cell search operation. For a detailed discussion of the basic problems in particle tracing, see [5] and [13].

3 Related Work

Presently, many systems are available for steady flow visualization. However, most of these systems only provide instantaneous visualization of the flow data. For example, instantaneous streamlines, isosurfaces, and slicing planes. Several effective techniques were recently developed for interactive interrogation of instantaneous flow fields. Some of these techniques are described in [6,8,11]. To date, there are very few particle tracing systems that can generate streaklines using a large number of time steps from unsteady flow fields. Two of these are Virtual Wind Tunnel (VWT) [3] and pV3 [7]. VWT provides interactive visualization of particle traces in a virtual environment using a stereo head-tracked display and a data glove. Although VWT is an effective interactive tool for unsteady flow visualization, it requires a preprocessing of the flow data, and the number of time steps that the user can visualize is determined by the memory size of the system. pV3 allows interactive animation of unsteady flow data by looping through an input file that contains the names of the flow data files. This is similar to an interactive scripting approach. pV3 does not save the visualization results, hence, playback is not supported and re-calculation is required to repeat the animation.

4 Requirements

It is common to generate several thousand time steps of flow data in a CFD simulation; however, it is presently impossible to visualize flow data from all these time steps at one time. Scientists sometimes use one of the following approaches: (1) visualize the data at some snapshots in time or (2) save every nth time step of the data and then visualize the subset of data. Regardless of the approach used, there are usually hundreds of time steps that need to be visualized [10]. A requirement for UFAT is that it must be able to compute streaklines from a large number of time steps.

A complex grid usually consists of several grid blocks. For some grids, one or more grid blocks may move as a function of time, a characteristic of grids with rigid-body motion. Moving grids are commonly used in pitching airfoils, oscillating flaps, rotating turbine fans of combustion engines, and rotating helicopter blades. Another requirement for UFAT is that

it must be able to compute particle traces in unsteady flow with moving grids.

The ability to visualize a scalar quantity in the flow data can be crucial for some flow analysis. Quantities that are commonly computed are temperature, pressure, mach number, and density. A requirement for UFAT is that it must assign a color to each particle based on the value of a specified quantity sampled at the particle's location. The color of the particle can also be based on its position, the time at which it was released, or the seed location where it was released.

Interactive visualization of large time-dependent flow fields is difficult or nearly impossible due to the data size. Sometimes, a scripting approach is used to save the visualization results from each time step, and the visualization results are then played back at a later time. In some visualization systems, interactive visualization is feasible by using one of the following approaches: (1) preprocess the data so that a number of time steps can be stored in memory or (2) sample the flow data at a lower resolution so that the data can be stored in memory. By storing the flow data in memory, the data can be interactively visualized. The latter approach is usually not desirable because the accuracy of the flow data is lost when the data is sampled at a lower resolution. With either approach, the size of the physical memory dictates how much flow data can be visualized interactively. Although these two approaches can provide interactive visualization, important features may not be detected because of the reduced representation of the flow data. Using a scripting approach, the entire flow data can be analyzed. Furthermore, once the visualization results have been saved, the scientist can play back the results repeatedly without any additional computation. Visualization playback is another requirement for UFAT.

5 Unsteady Flow Analysis Toolkit

UFAT was developed to compute streaklines using a large number of time steps in 3D unsteady flow fields. It handles single and multi- block curvilinear grids, and the grid may have rigid-body motion. Particles are released continuously from the specified seed locations at each time step. The particles are advected through all time steps until they leave the grid domain. UFAT saves the current positions of the particles at each time step; thus, the particle traces can be played back at a later time. Particles are colored according to a scalar quantity. The quantity may be a physical quantity of the flow (e.g. pressure, temperature, and density), a position coordinate (x, y, or z) of the particle, the time at which the particle was released, or the seed location where the particle was released. UFAT uses an adaptive-time integration scheme to advect the particles in the physical coordinate space. The integration scheme can be of second or fourth order. UFAT also allows particles to be traced along the grid surface. This type of particle trace simulates oil flow on a surface. Sometimes, the available disk on a system may not be able to store all time steps of flow data. UFAT provides a save/restore option so that particle tracing can be performed in several run sessions. This allows particle traces to be computed from many time steps without requiring all time steps to be online at one time.

5.1 Data Structure

In order to advect particles through all time steps, UFAT stores the two most recent time steps of the flow data in memory. Particles are successively advected from the current time step to the next time step. The particle traces are stored in a two-dimensional array of size $N_s \times N_t$, where N_s is the number of seed locations and N_t is the number of time steps. Each entry in the array is a structure that contains the physical and computational coordinates of the particle and the time at which the particle was released from the seed location. There is also an array of size N_s that stores the number of particles in each trace. Let $Trace_Length[s]$ denote the number of particles in trace s, and it is initialized to zero. At each time step, a new particle is released from the seed location and $Trace_Length[s]$ is incremented by one. When a particle in trace s leaves the grid domain, $Trace_Length[s]$ is decremented by one.

5.2 Algorithm

This section outlines the particle tracing algorithm in UFAT. The following procedures in the algorithm are described: `Step_Through_Time()`, `Advect_Trace()`, and `Advect_Particle()`. Procedure `Step_Through_Time()` steps through all time steps in the given flow data and calls `Advect_Trace()`. The main task of procedure `Advect_Trace()` is to advect the active particles in all traces from the current time step to the next time step. The actual particle integration is performed in procedure `Advect_Particle()`. For brevity, let *current_time* denote the current time step and *next_time* denote the next time step. For each procedure, a description is given followed by the pseudocode of the procedure.

Procedure `Step_Through_Time()` begins by loading the first two time steps of the flow and grid data

into memory. If the grid is fixed, then only one grid is loaded into memory. Then, it steps through all time steps in the flow data. For each time step, the following tasks are performed: (1) Call procedure **Advect_Trace()** to advect particles in every trace from *current_time* to *next_time*. (2) Write the current particle traces to the trace file. A frame marker is also written to denote the end of each time step. (3) Read the next time step's flow. If the grid is moving in time, then read the next time step's grid. The pseudocode for this procedure is given below:

```
Procedure Step_Through_Time()
Read first two time steps of flow and grid data
For t = 1 to N_t - 1 do
    Advect_Trace(t, t + 1)
    Write current traces to the trace file
    Read the next time step's flow data
    If moving grid then
       Read the next time step's grid
End for
```

Procedure **Advect_Trace()** advects particles in every trace from *current_time* to *next_time*. The procedure performs the following steps for each trace: (1) Copy all particles in the trace to a working trace array w. (2) Call procedure **Advect_Particle()** to advect each particle in the working trace w from *current_time* to *next_time*. If the particle is inside the grid domain D after the advection, then the particle is saved to the trace. Otherwise, the particle is considered to be inactive and it is discarded. (3) Release a new particle from the trace's seed location and save the particle in the trace. The pseudocode for this procedure is as follows:

```
Procedure Advect_Trace(current_time, next_time)
For s = 1 to N_s do
    Copy trace s to working trace w
    W_Length = Trace_Length[s]
    Remove all particles in trace s
    Trace_Length[s] = 0
    For i = 1 to W_Length do
        p = the ith particle in working trace w
        Advect_Particle(current_time, next_time, p)
        If p ∈ D then
           Store p in trace s
           Trace_Length[s] = Trace_Length[s] + 1
        End if
    End for
    Release a new particle from seed s and
        store it in trace s
    Trace_Length[s] = Trace_Length[s] + 1
End for
```

Procedure **Advect_Particle()** advects the given particle p from *current_time* to *next_time*. The pseudocode shown below uses the second-order Runge-Kutta integration scheme given in Equation (2) and is based on a predictor-corrector algorithm used in PLOT3D [5]. The flow data is only given at some number of time steps. If $t \neq t_i$ for $i = 1, \ldots, N_t$, then an interpolation in time is performed. Since the velocity is known only at discrete points in the grid, when p does not coincide with a grid point, a trilinear interpolation in physical space is also performed. Following are the steps in procedure **Advect_Particle()**: (1) Initialize t to *current_time*. The variable t is incremented at each advection and the procedure exits when t = *next_time* or when the particle has left the grid domain D. (2) Interpolate the velocity $\vec{V}$ at p. (3) Compute the time increment h, where $h = c/max(\vec{V})$ and c is a fraction of the grid cell that each particle must take inside the cell. The constant c can be considered as a normalized stepsize and $0 < c \leq 1$. For example, if the particle must traverse five steps in a cell, then let $c = 0.2$. (4) Increment t by h. (5) Compute the predictor p^*. (6) Interpolate the velocity $\vec{V}^*$ at p^*. (7) Compute the corrector, which is the position of p after the advection. Below is the pseudocode for procedure **Advect_Particle()**.

```
Procedure Advect_Particle(current_time, next_time, p)
t = current_time
While (t < next_time AND p ∈ D ) do
      V⃗ = Interpolate_Velocity(p, t, current_time,
        next_time)
Adjust:
      h = c/max(V⃗)
      If (t + h > next_time) h = next_time - t
      t = t + h
{ Predictor step }
      p* = p + h * V⃗
      V⃗* = Interpolate_Velocity(p*, t + h,
         current_time, next_time)
{ Adaptive stepsizing }
      V⃗_total = (V⃗ + V⃗*)/2
      If (h * max(V⃗_total) > c) then
         V⃗ = V⃗_total
         t = t - h
         Goto Adjust
      Endif
{ Corrector step }
      p = p + h * (V⃗ + V⃗*)/2
End while
```

Although the pseudocode shown above only provides a second-order integration scheme, a fourth-order integration scheme has also been implemented in

UFAT. Procedure `Interpolate_Velocity()`, which is not shown, interpolates velocity in time followed by a trilinear interpolation in the physical space. When a new position for p is computed, a cell search step is performed to determine the grid cell that p lies in (see Section 2).

5.3 Animation

The most effective method to view streaklines is to animate the particle traces. UFAT saves streaklines at each given time step to a trace file, which can then be animated with a visualization system. Although particles may traverse in non-uniform time steps, the positions of the particles are sampled at uniform time steps (i.e. the given time steps). The particle trace file contains basic graphics primitives such as points and lines. These basic primitives can be written in a format so that they can be easily read by other visualization systems such as AVS, IRIS Explorer, and FAST [2]. The only requirement for the visualization system is that it must be able to animate the streaklines through a given sequence of time steps.

6 Distributed Visualization

It is common that an unsteady flow data set is too large to be stored locally on a graphics workstation. The flow data set is often stored on a remote system with a large disk capacity. The computation is then performed on the remote system while the results are sent over the network to the graphics workstation for interactive visualization. The data transfer rate on the network must be fast enough so that the image on the graphics workstation can be updated at least 15 frames per second for a reasonable animation. The size of the particle traces at each time step is relatively small compared to the size of the grid file. Thus, the particle traces at each time step can be sent over the network in a reasonably short period of time. If the grid is moving in time, then a new grid must also be sent over the network at each time step. It may take several seconds to transfer a grid consisting of several million grid points, depending on the speed of the network. Hence, distributed flow visualization is practical if the data transfer rate of the network is fast enough to handle the amount of data that will be sent over the network.

7 Results

This section shows some streaklines that were computed by UFAT for three unsteady flow data sets. Although the examples shown in this section are only from CFD applications, other applications with time-dependent flow data can easily use UFAT to generate streaklines. The input data must consist of the grid geometry and the flow quantities sampled at the grid points. Currently, UFAT only supports curvilinear grids.

The first data set is a clipped Delta Wing with control surfaces, which oscillate at a frequency of eight Hertz (Hz) and with an amplitude of 6.65 degrees. Each oscillation cycle consists of 5,000 time steps. For visualization purpose every 50th time step is saved, for a total of 100 time steps per cycle. The clipped Delta Wing grid consists of 250 thousand points in seven blocks. For this CFD simulation, the scientists evaluated a new zoning method called "virtual zones," which is used for grids with time-varying boundary conditions. Virtual zones simplify the grid generation problem for complex geometries and for time-dependent geometries [9]. Figure 1 shows the seed locations, which are colored by position, at the leading edge of the clipped Delta Wing. Figure 2 shows streaklines at time step 7, where the control surfaces have deflected 2 degrees up. The evenly spaced particle traces (colored in cyan) near the center of the wing indicates that the flow is relatively steady in that region. However, the flow is very turbulent near the tip of the wing. Figure 3 shows the streaklines after the control surfaces have completed one oscillation. At this time, the control surfaces have deflected 5 degrees up. This figure shows that some particles (colored in red), which were released from the outer part of the leading edge of the wing, have moved toward the tip of the wing due to the control surfaces. This behavior can be seen clearly in an animation of the streaklines.

The second data set is an arrow wing configuration of a supersonic transport in transonic regime. Transonic flutter is known as a design problem on this configuration. Scientists want to develop a computational tool to examine the influence of control surface oscillations on the lift of the transport for the suppression of the flutter. The arrow wing grid consists of approximately one million points in four blocks. The control surfaces oscillate at a frequency of 15 Hz and with an amplitude of 8 degrees. Figure 4 shows streaklines surrounding the transport at time steps 25 and 175. The particles are colored by their seed locations, where the particles were released. It can be seen that there is vortical separation from the leading edges of the wing. From the simulation, it was found that the symmetric oscillation produces higher lift than the anti-symmetric oscillation [12].

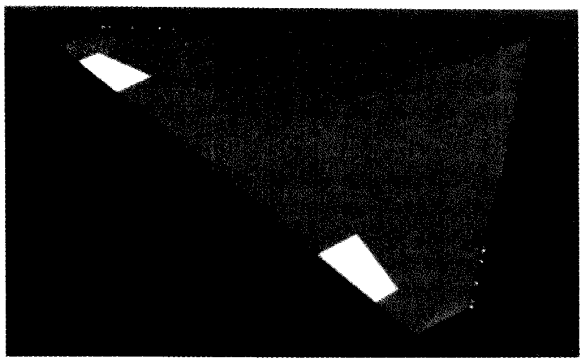

Figure 1. Seed locations on the leading edge of the clipped Delta Wing. Particles are colored by position.

Figure 2. Streaklines at time step 7. The control surfaces have deflected up by 2 degrees.

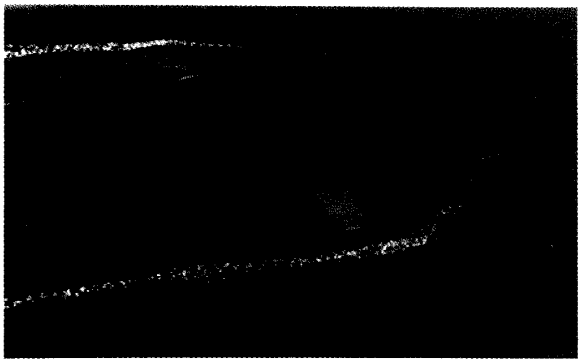

Figure 3. Streaklines at time step 112. The control surfaces have deflected up by 5 degrees.

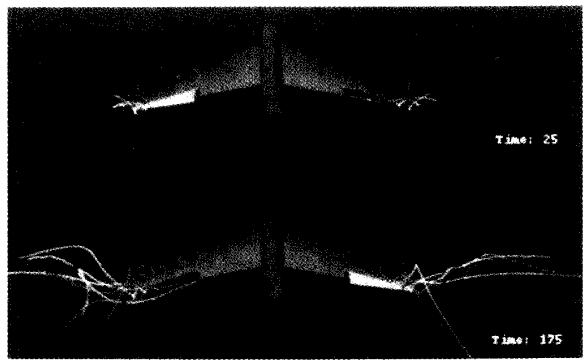

Figure 4. Streaklines from a supersonic transport at time steps 25 and 127.

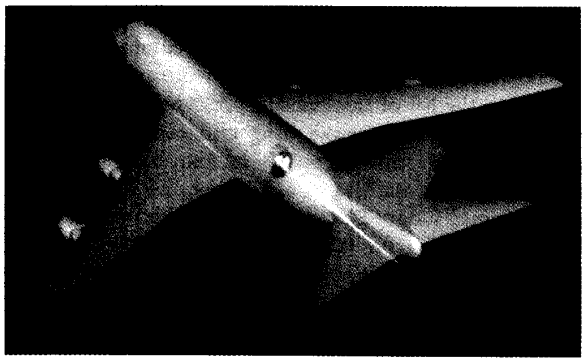

Figure 5. Streaklines surrounding the cavity of a modified Boeing 747SP, which is an airborne observatory.

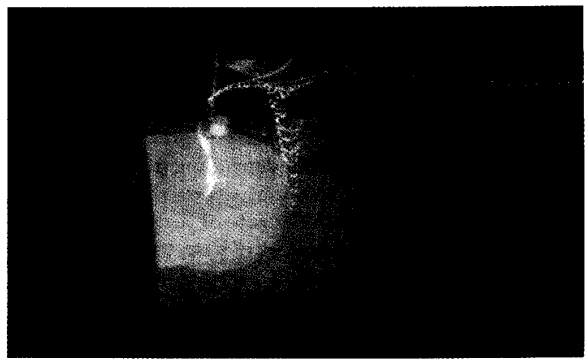

Figure 6. A close-up view of streaklines near the telescope (colored in cyan) inside the cavity of the observatory.

(See color plates, page CP-29.)

The third data set is the proposed airborne observatory known as the Stratospheric Observatory For Infrared Astronomy (SOFIA). SOFIA is a modified Boeing 747SP transport with a large cavity that holds a three-meter class telescope. The CFD scientists want to assess the safety and optical performance of a large cavity in the 747SP [1]. SOFIA would be the successor to the Kuiper Airborne Observatory (KAO), which is the only aircraft in the world that currently provides this type of infrared observing capability. SOFIA consists of approximately four million points in 41 grids. A total of 50 time steps were saved for the visualization. Figure 5 shows streaklines surrounding the SOFIA airborne observatory at time step 40. In the figure, the telescope (partially visible) inside the cavity of the jet is colored in cyan. The particles are colored by the time of their release from a rake positioned in the aperture of the cavity. Blue represents the earliest time and orange represents the most recent time. Figure 6 shows a close-up view of the telescope without the aircraft body. The floating object (colored in gray) above the telescope is the secondary mirror of the telescope. The cavity is represented by the semitransparent surface enclosing the telescope. Note that some particles are trapped inside the cavity, while some have escaped and passed the empennage.

8 Performance

The performance of UFAT depends on three factors: (1) the grid size, (2) the number of time steps, and (3) the number of seed locations. At each time step, UFAT reads the flow data file (and the grid file if the grid is moving in time). Depending on the disk I/O rate and the grid size, it could take from several seconds up to several minutes to read the flow and grid files at each time step. For example, if the disk I/O rate is 10 megabytes per second, then it would take approximately 1.2 seconds to read a grid file with one million grid points, assuming that there are three physical coordinates (x, y, and z) for each grid point. If the flow data file (solution file) contains five scalar quantities, then it would take approximately 2.0 seconds to read the file. Thus, it would require a total of 3.2 seconds per time step to read the grid and solution data. The number of particles that UFAT advects at each time step increases linearly. If there are 100 seed locations and 1,000 time steps, then the maximum number of particles that UFAT can advect at time step 1,000 is 100,000 particles. Using the clipped Delta Wing as an example, it took approximately 2.3 seconds to read a grid file and 3.0 seconds to read a solution file at each time step on a Silicon Graphics 320 VGX graphics workstation. The size of the grid file is four megabytes and the solution file is five megabytes. It took approximately 21 minutes to compute streaklines from 100 time steps and with 36 seed locations using a single 33-megahertz processor on the VGX graphics workstation. This includes the time to read the grid and solution files at each time step. The size of the trace file generated by UFAT is 2.5 megabytes.

9 Future Work

A disadvantage of the current version of UFAT is that it performs particle tracing sequentially. Particle tracing is an "embarrassingly" parallel application. It would be ideal to take advantage of multiple processors to perform particle tracing in parallel since each particle trace can be computed independently. A parallel version of UFAT has been developed on the Cray C90, Convex C3240, and SGI systems. The initial results indicate that the performance can be improved by several factors, depending on the number of processors used. Another enhancement that is currently being investigated is how to distribute the particle tracing task to a cluster of heterogeneous systems using a message passing library. An issue to be worked out is how to minimize the amount of data that each system needs for particle trace computation. The goal is to have a distributed parallel version of UFAT that would provide interactive particle tracing in large-scale unsteady flow fields.

Acknowledgments

The flow data sets were provided by Chris Atwood, Goetz Klopfer, Steve Klotz, and Shigeru Obayashi. This work was supported by NASA under contract NAS 2-12961.

References

[1] Atwood, C. and van Dalsem, W., Flowfield Simulation about the Stratospheric Observatory for Infrared Astronomy, *AIAA Journal of Aircraft*, Monterey, California, September 1993, pp. 719-727.

[2] Bancroft, G., Merritt, F., Plessel, T., Kelaita, P., McCabe, K., and Globus, A., FAST: A Multi-Processed Environment for Visualization of Computational Fluid Dynamics, in: A. Kaufman, ed., *Proceedings of Visualization '90,* San Francisco, California, October 1990, pp. 14-27.

[3] Bryson, S. and Levit, C., The Virtual Wind Tunnel, *IEEE Computer Graphics & Applications,* Vol. 12, No. 4, July 1992, pp. 25-34.

[4] Buning, P., Sources of error in the graphical analysis of CFD results, *Journal of Scientific Computing,* Vol. 3, No. 2, 1988, pp. 149-164.

[5] Buning, P. and Steger, J., Graphics and Flow Visualization in Computational Fluid Dynamics, *7th Computational Fluid Dynamics Conference,* Cincinnati, Ohio, July 1985, AIAA 85-1507.

[6] de Leeuw, W. and van Wijk, J., A Probe for Local Flow Field Visualization, in: G. Nielson and D. Bergeron, eds., *Proceedings of Visualization '93,* San Jose, California, October 1993, pp. 39-45.

[7] Haimes, R., pV3: A Distributed System for Large-Scale Unsteady CFD Visualization, *32nd AIAA Aerospace Sciences Meeting and Exhibit,* Reno, Nevada, January 1994.

[8] Hin, A. and Post, F., Visualization of Turbulent Flow with Particles, in: G. Nielson and D. Bergeron, eds., *Proceedings of Visualization '93,* San Jose, California, October 1993, pp. 46-52.

[9] Klopfer, G. and Obayashi, S., Virtual Zone Navier-Stokes Computations for Oscillating Control Surfaces, *11th Computational Fluid Dynamics Conference,* Orlando, Florida, July 1993, AIAA 93-3363-CP.

[10] Lane, D., Visualization of Time-Dependent Flow Fields, in: G. Nielson and D. Bergeron, eds., *Proceedings of Visualization '93,* San Jose, California, October 1993, pp. 32-38.

[11] Max, N., Becker, B., and Crawfis, R., Flow Volumes for Interactive Vector Field Visualization, in: G. Nielson and D. Bergeron, eds., *Proceedings of Visualization '93,* San Jose, California, October 1993, pp. 19-24.

[12] Obayashi, S., Chui, I., and Guruswamy, G., Navier-Stokes Computations on Full-Span Wing-Body Configuration with Oscillating Control Surfaces, *AIAA Atmospheric Flight Mechanics Conference,* August 1993, AIAA-93-3687.

[13] Post, F. and van Walsum, T., Fluid Flow Visualization, in: H. Hagen, H. Mueller, and G. Nielson, eds., Focus on Scientific Visualization, Springer, Berlin, 1993, pp. 1-40.

[14] Reeves, W., Particle Systems - A Technique for Modeling a Class of Fuzzy Objects, ACM Transaction on Graphics, Vol. 2, 1983, pp. 91-108.

[15] Schlichting, H., Boundary Layer-Theory, McGraw Hill, New York, 1979.