Using Residence Time for the Extraction of Recirculation Regions

Robert Haimes*
Department of Aeronautics and Astronautics
Massachusetts Institute of Technology, Cambridge, MA 02139
haimes@orville.mit.edu

This paper introduces the concept of residence-time, from the Eulerian view point, in a rigorous manner. The equations for various flow regimes are derived and a numerical solver is introduced based on Lax-Wendroff integration. An implementation is discussed that allows the coupling of this solver to any explicit CFD code. Examples of this concept are shown for extracting recirculation regions by segregating old fluid from fluid that has not been in the simulation for much time. The comparison of iso-surfaces generated using this procedure and separation surfaces are examined.

Introduction

In the past, feature extraction and identification were interesting concepts, but not required to understand the underlying physics of a steady flow field. This is because the results of the more traditional tools like iso-surfaces, cuts and streamlines were more interactive and easily abstracted so they could be represented to the investigator. These tools worked and properly conveyed the collected information at the expense of much interaction. For unsteady flow-fields, the investigator does not have the luxury of spending time scanning only one "snap-shot" of the simulation. Automated assistance is required in pointing out areas of potential interest contained within the flow.

Automated feature detection and identification procedures are being developed for the examination of 3D transient simulations. The final expression of this work will be a software tool-kit that allows for the post-processing or co-processing visualization of Computational Fluid Dynamics results. The captured features will be constructed and displayed in a manner that physically makes sense to the investigator. Also, these techniques will allow "off-line" procedures (like grid adaptation and design optimization) to use the physics found in the flow-field to perform the task.

This paper discusses a technique that locates regions of recirculation in both steady-state and transient solutions.

Separation and Recirculation

Flow separation represents interesting, and sometimes important, features found in many flow regimes. For combustion, swirling flow is used to enhance mixing but can also cause isolated regions of flow that do not quickly leave the system. These regions may be high-temperature and therefore can have a negative impact on the lifetime of the unit. Similarly, in turbomachinery, separated flows are associated with extremely hot regions where high-speed flow exiting the combustor has stagnated. These hot spots are undesirable since the allowable operating stress of the turbine blades is closely related to temperature.

In external aerodynamics, the adverse pressure gradient on the wing upper surface can lead to separated flow causing a drastic loss of lift (or stalling) and a significant increase in pressure drag.

The importance of separated flow motivates the development of a tool which can automatically locate these regions. Ideally for steady-state flows, this tool can operate directly on the vector field imported from the flow solution. It is desirable for transient simulations to require only a single time slice of the field data. Needing multiple time-steps can quickly overwhelm the memory resources available for the compute.

Individuals investigating the results of CFD simulations have historically seeded streamlines (in steady-state flows) to determine where there are regions of recirculation. With a numerically accurate streamline integration scheme, recirculation is found when a streamline is "trapped" in the flow-field. In this case the streamline, going up-stream or down-stream from a point within the region, does not leave the domain. The boundary of this region is a surface – the separation surface. There are a number of problems in automating this interactive procedure:

- Locating the region.
  Automatically seeding streamline to find these regions is difficult, indeed. Any streamline started outside the region will not enter. Therefore, one can not use the in-flow of the domain as a starting point. One would need to seed large numbers of streamlines so that every cell in the mesh is touched to insure that the region(s) are found.

*Principal Research Engineer

- Cost.
  Clearly, the cost of such a procedure would be prohibitive. There is also the issue of stopping the integration once a point inside the region is found.

- Constructing the surface.
  It is not obvious how one takes the outer "husk" of the trapped streamlines and then construct the separation surface.

The discussion up to this point assumes steady-state flow. It is unclear how a recirculation region should be defined in unsteady flows since a region that appears to be recirculating at an instantaneous time slice might actually be moving with the flow as time progresses. It is well known that examining streamlines to understand transient flows can be misleading. Streaklines need to be used.

Helman and Hesselink[1,2] have developed a visualization scheme for generating separation surfaces using only the solution's vector field. The scheme starts by finding the critical points on the surface of the object. Streamlines are integrated along the principal directions of certain classes of critical points and then linked back to the critical points to produce a 2-D skeleton of the flow topology near the object. Streamlines are integrated out to the external flow starting from points along certain curves in the skeleton. These streamlines are then tessellated to generate the separation surfaces. With this approach, difficulties might be encountered in integrating streamlines from critical points, and also in finding separated regions that are not attached to an object.

Sujudi[3] attempted an eigenanalysis technique looking for topology where the flow is diverging locally from a plane. This was done after the success of finding vortices by looking for swirling flow, also see Sujudi.[4] This technique was determined to be unreliable. Critical point theory only provides a single classification based on the strongest local topology. In most separated flow regimes there are areas where the swirl component is much greater than the diverging topology. This can overwhelm the ability to locate the separation surface. But, Kenwright[5] has had some success in using 2D critical points for finding surface separation and attachment lines. Both the vortex core finder[4,6] and the finder of surface separation and attachment line work well in transient regimes.

This paper discusses a method that attempts to stay within the streamline/streakline definition of a separated region but applies a transformation from Lagrangian (moving with the massless particle) to an Eulerian point of view (fixed within a grid and watching the fluid flow past). The concept is *residence-time.* Essentially, one computes the amount of time the fluid has been in (or resident within) the domain. Residence-time zero is defined as the time when the simulation starts. Most of the fluid within a separation region stays within that region for a considerable amount of time. Thus, a common feature of separation region is that the residence-time of the fluid within is much larger than that of the surrounding fluid. An iso-surface can then be used to distinguish this region. The value of the iso-surface can be selected knowing the characteristic time for the system. Therefore, residence-time can easily produce the separation surface for either steady-state or transient simulations.

This can be thought of as a streakline process where, periodically at the in-flow, particles are seeded and marked with their start-time. At some later simulation time the particles that have aged the most are segregated and marked. The advantage of residence-time is that you get complete grid coverage and the surface generation is trivial.

The visualization test-bed used in this paper is **pV3**,[7,8] a distributed system developed at MIT. **pV3** is ideal for this work, in that, it is designed for co-processing. Co-processing allows the investigator to visualize the data as it is being computed by the solver. Distributed computing decomposes the computational domain into 2 or more sub-domains, which can be processed across a network of workstation(s) and other types of compute servers. The algorithm used in computing residence-time has been developed to co-exist with the parallel capability of **pV3**.

Theory

The residence-time of a volume of inviscid fluid is defined by

$$\frac{\mathbf{D}\mathcal{T}}{\mathbf{Dt}} = 1 \tag{1}$$

where $\mathcal{T}$ denotes residence-time. This is similar to conservation of mass, but with a source term. It should be noted that this author has not seen any references to this definition in the open literature.

Since $\frac{\mathbf{D}}{\mathbf{Dt}} \equiv \frac{\partial}{\partial t} + \vec{u} \cdot \nabla$, then Equation (1) becomes

$$\frac{\partial \mathcal{T}}{\partial t} + \vec{u} \cdot \nabla \mathcal{T} = 1 \tag{2}$$

where $\vec{u}$ is the velocity vector.

Since the time when the residence-time calculation starts is defined as time zero, then initial the condition is

$$\mathcal{T}(x, y, z) = 0. \tag{3}$$

At in-flow boundaries, new fluid is entering. By definition, this fluid has zero residence-time. Therefore, the boundary condition is

$$\mathcal{T}(x, y, z) = 0 \qquad at\ in-flow. \tag{4}$$

To obtain the conservative form of Equation (2) for incompressible flows, this can be rewritten as

$$\frac{\partial \mathcal{T}}{\partial t} + \vec{u} \cdot \nabla \mathcal{T} + \mathcal{T} \nabla \cdot \vec{u} = 1 + \mathcal{T} \nabla \cdot \vec{u}$$

or

$$\frac{\partial \mathcal{T}}{\partial t} + \nabla \cdot (\mathcal{T}\vec{u}) = 1 + \mathcal{T}\nabla \cdot \vec{u}.$$

And since $\nabla \cdot \vec{u} = 0$ for incompressible flows, then

$$\frac{\partial \mathcal{T}}{\partial t} + \nabla \cdot (\mathcal{T}\vec{u}) = 1. \tag{5}$$

Equation (5) reflects the residence-time in an analogous manner to streaklines. Here the result is only a function of the velocity field. Additional realism may be applied to the formulation. The conservative form for compressible flows can be obtained by rewriting Equation (2) as

$$\rho\frac{\partial \mathcal{T}}{\partial t} + \rho\vec{u} \cdot \nabla\mathcal{T} = \rho$$

where ρ denotes density. And since the conservation of mass equation for compressible flows is

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho\vec{u}) = 0$$

then

$$\rho\frac{\partial \mathcal{T}}{\partial t} + \rho\vec{u} \cdot \nabla\mathcal{T} + \mathcal{T}\left[\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho\vec{u})\right] = \rho$$

or more simply

$$\frac{\partial (\rho\mathcal{T})}{\partial t} + \nabla \cdot (\rho\mathcal{T}\vec{u}) = \rho. \tag{6}$$

The effect of viscosity on $\mathcal{T}$ is the same as its effect on velocity in that the same mechanism is at work. This is justified from the statistical mechanics viewpoint. At the molecular level, viscous action mixes individual molecules (each with it's own residence-time), so the average local residence-time is effected by viscosity.[9] The viscous term for Equation (6) mimics the viscous term in the conservation of momentum equation of the Navier-Stokes equations. Thus, the residence-time equation for a viscous compressible flow is

$$\frac{\partial (\rho\mathcal{T})}{\partial t} + \nabla \cdot (\rho\mathcal{T}\vec{u}) = \rho + \nabla \cdot (\mu\nabla\mathcal{T})$$

or

$$\frac{\partial (\rho\mathcal{T})}{\partial t} + \nabla \cdot (\rho\mathcal{T}\vec{u} - \mu\nabla\mathcal{T}) = \rho \tag{7}$$

where μ is the total viscosity, which accounts for both laminar and turbulent components.

For the special case where the flow has both a constant viscosity and density, Equation (7) reduces to

$$\frac{\partial \mathcal{T}}{\partial t} + \nabla \cdot (\mathcal{T}\vec{u}) = 1 + \kappa\nabla^2\mathcal{T}$$

and further to

$$\frac{\partial \mathcal{T}}{\partial t} + \nabla \cdot (\mathcal{T}\vec{u} - \kappa\nabla\mathcal{T}) = 1 \tag{8}$$

where $\kappa = \frac{\mu}{\rho}$ which is a constant.

All the conservative forms of the residence-time equations [Equations (5), (6), (7), and (8)] can be expressed as

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} + \frac{\partial H}{\partial z} = Q \tag{9}$$

For incompressible inviscid flow

$$U = \mathcal{T}, \quad F = \mathcal{T}u, \quad G = \mathcal{T}v, \quad H = \mathcal{T}w, \quad Q = 1,$$

where u, v, and w are the components of $\vec{u}$. For compressible inviscid flow

$$\begin{aligned} U = \rho\mathcal{T}, \qquad F = {}& \rho\mathcal{T}u, \quad G = \rho\mathcal{T}v, \\ H = {}& \rho\mathcal{T}w, \quad Q = \rho, \end{aligned}$$

for compressible viscous flow

$$\begin{aligned} U = \rho\mathcal{T}, \qquad F = {}& \rho\mathcal{T}u - \mu\frac{\partial \mathcal{T}}{\partial x}, \quad G = \rho\mathcal{T}v - \mu\frac{\partial \mathcal{T}}{\partial y}, \\ H = {}& \rho\mathcal{T}w - \mu\frac{\partial \mathcal{T}}{\partial z}, \quad Q = \rho, \end{aligned}$$

and for a flow with constant viscosity and density

$$\begin{aligned} U = \mathcal{T}, \qquad F = {}& \mathcal{T}u - \kappa\frac{\partial \mathcal{T}}{\partial x}, \quad G = \mathcal{T}v - \kappa\frac{\partial \mathcal{T}}{\partial y}, \\ H = {}& \mathcal{T}w - \kappa\frac{\partial \mathcal{T}}{\partial z}, \quad Q = 1. \end{aligned}$$

Residence-time Integration

The clear disadvantage of using residence-time to extract regions of recirculation is that a Partial Differential Equation (PDE) needs to be solved. Equation (9) has a form similar to the conservative formulation of the Euler equations. This similarity enables the use of integration schemes developed for either the Euler or Navier-Stokes equations. It should also be noted that the coupling is looser than turbulence models. $\mathcal{T}$ is a function of $\vec{u}$ and optionally ρ and μ but does not feedback to the Euler or Navier-Stokes equations.

Ideally the solver writer would include another entry ($\mathcal{T}$) to the state vector. The time-step requirement for residence-time is less restrictive than either the Euler or Navier-Stokes equations because of the lack of acoustic waves so time-marching would not be effected.

In the case where modifying the solver can not be considered (and in pure post-processing applications) a residence-time solver is required. The scheme discussed in the rest of this paper is explicit, operates on a cell-by-cell manner, and can therefore take advantage of **pV3**'s parallel capability. Coupling to any implicit solver would require more care do to the time-step restrictions.

In producing a tool for general use it is important to consider the design goals. In the case of selecting an integration scheme for solving the residence-time PDE the following must be considered:

- Spatial Accuracy. The spatial accuracy must be at least as good as the solver.
- Temporal Accuracy. The temporal accuracy should be consistent with the solver and be at least second order accurate so it can be used in unsteady simulations.
- Numerical Dissipation. The dissipation produced by the scheme must be a minimum so that the sharp gradients in residence-time generated at the separation surfaces are not smeared.
- Node Based. The function of iso-surfacing is done where the value is at nodes that support the cells of the mesh. The scheme selected must be able to accurately place $\mathcal{T}$ at the nodes and with minimal averaging.
- Support General Discretization. **pV3** supports structured blocks, disjoint tetrahedra, pyramids, prisms and hexahedra as either a homogenous or heterogeneous collection. The scheme selected should be able to deal with general meshing.

Lax-Wendroff

The explicit time-marching algorithm of the Lax-Wendroff type is a good choice to solve Equation (9). The Lax-Wendroff scheme is both second order in space and time. Little numerical dissipation is generated by the method and only small amounts of numerical smoothing are required under favorable conditions. The result is node based and the scheme can be cast for the general discretization supported by **pV3**.

The integrator developed is similar to that used by Saxer[10] to solve the Euler equations for turbomachinery stator/rotor flow. The basic integration scheme was introduced by Ni,[11] recast by Hall,[12] and then extended by Ni and Bogoian[13] to 3-D. Saxer[10] then adapted the formulation to handle hexahedral unstructured grids. For this tool the complete suite of cell types are supported. Structured blocks are treated as a collection of disjoint hexahedra.

In solving the residence-time equation, it is assumed that the flow variables $\vec{u}$, and optionally ρ, and μ are known at all the nodes. The algorithm then computes the flux across each cell face by averaging the fluxes F, G, and H at the appropriate nodes. The flux residual is computed by adding the fluxes through the faces of the element, and then adding the source term for the cell. This residual is then distributed back to the nodes according to the Lax-Wendroff algorithm to evaluate the change to the residence-time.

In a Lax-Wendroff scheme there are 3 components; the first and second order terms as well as the source term. The major part of the first order term is cell based. For cell A:

$$\Delta U_A = -\frac{\Delta t}{V_A} \sum_{faces}^{Cell A} \left(\bar{F} S_x + \bar{G} S_y + \bar{H} S_z \right) + \Delta t Q.$$

The bar over F, G, and H denotes an average over the nodes associated with the cell face. S_x, S_y, and S_z are the projected areas on the yz, xz, and xy planes for the specific face. V_A is the volume of the cell and Δt is the time-step.

The second order term is node based and requires the dual of the mesh. The contribution is based upon what proportion of the intersection of the cell of interest and the pseudo-mesh cell centered on node i. Specifically the second order and source terms are:

$$\Delta U_{iA} = -\sum_{faces}^{dual} \left(\Delta F_A \tilde{S}_x + \Delta G_A \tilde{S}_y + \Delta H_A \tilde{S}_z \right) + \frac{V_A}{2} \Delta Q_A.$$

Where $\tilde{S}_x$, $\tilde{S}_y$ and $\tilde{S}_z$ are the projected areas on the yz, xz, and xy planes for the faces of the pseudo-mesh cell contained in cell A. For inviscid incompressible flow:

$$\begin{aligned}
\Delta F &= \bar{u}\Delta\mathcal{T} + \bar{\mathcal{T}}\Delta u \\
\Delta G &= \bar{v}\Delta\mathcal{T} + \bar{\mathcal{T}}\Delta v \\
\Delta H &= \bar{w}\Delta\mathcal{T} + \bar{\mathcal{T}}\Delta w \\
\Delta Q &= 0
\end{aligned}$$

for inviscid compressible flow

$$\begin{aligned}
\Delta F &= \bar{u}\Delta(\rho\mathcal{T}) + \overline{(\rho\mathcal{T})}\Delta u \\
\Delta G &= \bar{v}\Delta(\rho\mathcal{T}) + \overline{(\rho\mathcal{T})}\Delta v \\
\Delta H &= \bar{w}\Delta(\rho\mathcal{T}) + \overline{(\rho\mathcal{T})}\Delta w \\
\Delta Q &= \Delta\rho
\end{aligned}$$

for inviscid compressible flow

$$\begin{aligned}
\Delta F &= \bar{u}\Delta(\rho\mathcal{T}) + \overline{(\rho\mathcal{T})}\Delta u - \bar{\mu}\Delta\frac{d\mathcal{T}}{dx} - \overline{\frac{d\mathcal{T}}{dx}}\Delta\mu \\
\Delta G &= \bar{v}\Delta(\rho\mathcal{T}) + \overline{(\rho\mathcal{T})}\Delta v - \bar{\mu}\Delta\frac{d\mathcal{T}}{dy} - \overline{\frac{d\mathcal{T}}{dy}}\Delta\mu \\
\Delta H &= \bar{w}\Delta(\rho\mathcal{T}) + \overline{(\rho\mathcal{T})}\Delta w - \bar{\mu}\Delta\frac{d\mathcal{T}}{dz} - \overline{\frac{d\mathcal{T}}{dz}}\Delta\mu \\
\Delta Q &= \Delta\rho
\end{aligned}$$

and, for flow with constant density and viscosity

$$\begin{aligned}
\Delta F &= \bar{u}\Delta(\rho\mathcal{T}) + \overline{(\rho\mathcal{T})}\Delta u - \kappa\Delta\frac{d\mathcal{T}}{dx} \\
\Delta G &= \bar{v}\Delta(\rho\mathcal{T}) + \overline{(\rho\mathcal{T})}\Delta v - \kappa\Delta\frac{d\mathcal{T}}{dy} \\
\Delta H &= \bar{w}\Delta(\rho\mathcal{T}) + \overline{(\rho\mathcal{T})}\Delta w - \kappa\Delta\frac{d\mathcal{T}}{dz} \\
\Delta Q &= 0.
\end{aligned}$$

Except for quantities defined above, the subscript A denotes quantities evaluated at cell A, while the subscript i stands for average quantities at node i. Quantities marked with the over-bar are averages over the cell. Other quantities designated with the Δ operator are the change in that quantity with respect to time.

Finally, the complete contribution for node i in cell A is computed by:

$$\delta U_{iA} = \frac{1}{8}\frac{\Delta t}{V_i}\left\{\frac{V_A}{\Delta t}\Delta U_A + \Delta U_{iA}\right\}. \qquad (10)$$

The contributions to node i (from all cells that touch the node) are computed. The sum of these contributions define the change at node i:

$$\delta U_i = \sum_{J=1}^{cells} \delta U_{iJ}.$$

Boundary conditions

Three types of boundary conditions must be considered: inlet, outlet, and wall. At the inlet, or the in-flow boundary, new fluid enters the computational domain. By definition, the residence-time of this fluid is zero. Therefore, the condition at the inflow boundary for any node i at the inflow is:

$$U_i = 0.$$

At outlet boundaries, the simplest boundary condition has been used, which is to do nothing – extrapolate. This assumption is reasonable as long as the gradients of the flow quantities in the direction normal to the boundary are small. However, if the user determines a more elaborate boundary treatment would be more appropriate, a call-back can be supplied which overrides the standard method. The values of U and δU_i of all the nodes will be passed to this subroutine, and the necessary modifications/adjustments (be it for the outlet boundary nodes or any other regions of the flow) can made.

At a wall boundary with non-zero velocity, the scheme must ensure that there is no flux through the wall. A flux through the wall would either create or destroy residence-time producing a non-physical result.

The wall condition is accomplished by making a correction to the contribution of nodes that touch the wall. The correction is performed after the Lax-Wendroff changes δU have been applied. The contribution of cell A (where A touches the wall) to δU_{iA} is corrected as follows:

$$\delta U_{iA} + \frac{1}{8}\frac{\Delta t}{V_i}\left(\bar{F}S_x + \bar{G}S_y + \bar{H}S_z\right)_{face=wall}.$$

Numerical smoothing

As stated above, Lax-Wendroff was selected, in that, under some conditions little numerical dissipation is required. A fourth-difference smoothing operator is required to damp out saw-tooth oscillations emitted by the scheme in the presence of strong gradients in $\vec{u}$ (such as at shocks). But, a fourth-difference smoother alone is insufficient. In fact, it will worsen the stability of the solution. This has to do with the dispersive characteristics of Lax-Wendroff under large gradients. The integration scheme ***kicks-up*** oscillations on both sides of a discontinuity. This causes negative values of residence-time on one side of the discontinuity and values of $\mathcal{T}$ larger than the current simulation on the other.

To make matters worse, strong discontinuities are found at the bounds of any recirculation region. Inside the region, the values of $\mathcal{T}$ would continue to get larger and larger where outside the region the values remain constant (for steady-state flow).

Therefore, a mix of second-difference and fourth-difference smoothing is used. This mix is switched so that under high gradients only the second order smoother is applied and under quiescent conditions only the fourth-difference smoothing is used. The switch used is simply

$$\mathcal{S}_i = \frac{\sum_k^{edge-nodes}(U_k - U_i)}{\epsilon + \sum_k^{edge-nodes}\sigma_1\left|U_k - U_i\right| + \sigma_0\left|U_k\right|}$$

where $\mathcal{S}_i$ is the switch value for node i. This value must be bounded on the high side at 1.0. σ_0 and σ_1 are coefficients that are currently set to 0.05 and 1.0 respectively.

The fourth-difference smoothing operator, identical to the one used by Saxer,[10] has been constructed for the general discretization of volumes. The smoothing term is added to the right-hand-side of Equation (10) and has the form

$$-\sigma_4\nabla\cdot\left(\ell\nabla\left(\ell^2\nabla^2 U\right)\right)$$

where σ_4 is the fourth-difference smoothing coefficient and ℓ is a length comparable to the local grid size. In discrete form, the smoothing operator becomes

$$-\sigma_4\frac{1}{V_i}\sum_J^{cells} V_J\left(\bar{D}_J^2 - D_i^2\right)$$

where V_J is the volume of the cell. D_i^2 is a pseudo-Laplacian based on the all of the edge nodes surrounding node i. It is defined by Holmes and Connell[14] as

$$D_i^2 = \sum_k^{edge-nodes}\Psi_k\left(U_k - U_i\right)$$

where Ψ_k is a grid-dependent weight which determines the degree of dependence on the neighboring nodes. For details on how these weights are obtained, refer to Saxer.[10]

$\bar{D}_J^2$ is the discrete representation of a cell-averaged pseudo-Laplacian, and is

$$\bar{D}_J^2 = \frac{1}{\#nodes} \sum_k^{nodes} D_k^2.$$

The smoothing operator finally becomes

$$\begin{aligned} \delta U_i|_{smo} = \quad & \sigma_2 \mathcal{S}_i \tfrac{1}{V_i} \textstyle\sum_J^{cells} V_J \left(\bar{U}_J - U_i\right) \\ & -\sigma_4 (1-\mathcal{S}_i) \tfrac{1}{V_i} \textstyle\sum_J^{cells} V_J \left(\bar{D}_J^2 - D_i^2\right) \end{aligned}$$

where σ_2 is the second-difference smoothing coefficient given a value of between 0.001 and 0.01. σ_4 is the fourth-difference coefficient and is generally given a value of between 0.0001 and 0.005.

The Programming Interface

The residence-time integrator is part of a developing **F**eature e**X**traction software tool-kit: **FX**. The **FX** and the **pV3** programming interfaces are similar in form and coupling with the solver. Both require a call to an initialization routine that defines the spatial and temporal problem. Also, a call is required at the end of the iterative loop (when all of the flow variables have been updated – new values for $\vec{u}$, μ and ρ have been computed). These and the few other calls added to the solver and they have the prefix naming convention "FX_". This minimizes the changes to the solver and allows for easy removal during debugging.

The solver data is communicated through call-backs. These user-supplied routines are invoked by the "FX_" calls and their responsibility is to have the appropriate data arrays filled. The call-back routines are differentiated in that they do not contain the '_' after the "FX".

`FX_Init(iopt,knode,nhalo,ntets,npyra,nprism,nhexa, nblks,blocks,nhcell,nsurf,nbc,flags)`

This routine must be called before any other **FX** functions. It specifies the case to be run.
`int iopt` – the unsteady type (0-3).
`int knode` – number of non-block nodes.
`int nhalo` – number of halo nodes (decomposition).
`int ntets` – the number of tetrahedra.
`int npyra` – the number of pyramids.
`int nprism` – the number of prisms.
`int nhexa` – the number of hexahedra.
`int nblks` – number of structured blocks.
`int blocks` – the block definitions.
`int nhcell` – number of halo cells (decomposition).
`int nsurfa` – number of exterior surface facets.
`int nbc` – number of boundary conditions groups.
`int flags` – informs **FX** how to deal with memory.

`FX_RTParams(type, sm2, sm4, kappa)`

This routine must be called before any other residence-time functions.
`int type` – 0 to 3 for inviscid incompressible, viscous compressible, constant viscosity and density and inviscid compressible, respectively.
`float sm2` – (σ_2).
`float sm4` – (σ_4).
`float kappa` – κ, required for `type = 2` only.

`FX_Update(time)`

This subroutine must be called after the solver has updated the solution space. This is when all communication between any partitions is complete including the messages required to transmit the halo data.
`float time` – the current simulation time.

`FX_RTSolve()`

This routine must be called after `FX_Update` to integrate the residence-time equation for the time-step.

`FX_RTget(tau)`

This call retrieves the current value of $\mathcal{T}$ for all nodes. This can be placed in the scalar field **pV3** call-back, `pVScal`, to fill the residence-time scalar for visualization.
`float tau[]` – the current values of residence-time.

`FXcell(tets, pyra, prism, hexa, hcell)`

`FXcell` defines the volume discretization. It is not required for a grid that contains only structured blocks and no halo cells.
`int tets[ntets][4]` – the tetrahedra definitions.
`int pyra[npyra][5]` – the pyramid definitions.
`int prism[nprism][6]` – the prism definitions.
`int hexa[nhexa][8]` – the hexahedra definitions.
`int hcell[nhcell][9]` – the halo cell definitions.
  The cell type in `hcell[m][8]`:
  1 = tetrahedron (4 nodes indices in `hcell`)
  2 = pyramid (5 nodes indices in `hcell`)
  3 = prism (6 nodes indices in `hcell`)
  4 = hexahedron (8 nodes indices in `hcell`)

`FXsurface(nsurf, scel)`

This subroutine supplies **FX** with the surface data structure. This specifies that these are exposed facets and indicates the type of boundary condition to apply.
`int nsurf[nbc][2]` – the number of facets and type.
  `nsurf[m][1]` is the boundary condition type:
  1 = inflow
  2 = outflow
  3 = wall
  4 = wall (slip)
  5 = symmetry
  6 = nothing – extrapolate
`int scel[nsurfa][4]` – node indices for each facet.

`FXgrid(xyz, hxyz)`

This subroutine supplies **FX** with the grid coordinates for all of the nodes.
`float xyz[nNode][3]` – (x, y, z)-coordinates of grid nodes.

`float hxyz[nhalo][3]` – (x, y, z)-coordinates of halo grid nodes.

`FXvel(vel, hvel)`

This routine is always required. It defines $\vec{u}$ for the entire volume.
`float vel[nNode][3]` – Node based velocities.
`float hvel[nNode][3]` – Velocities for halo nodes.

`FXscal(type, s, hs)`

This subroutine supplies **FX** with the specified scalar field.
`int type` – Scalar type:
  $1 =$ density (ρ).
  $4 =$ total viscosity (μ).
`float s[nNode]` – Node based scalar.
`float hs[nNode]` – Scalar for halo nodes.

`FXmodifyRT(U, dU)`

This routine, if supplied, allows the modification of either δU_i and/or U_i directly. This is required for special boundary conditions, such as moving interfaces, periodics or internal boundaries due to domain decomposition.
`float U[nNode]` – Node based Us.
`float dU[nNode]` – Node based δUs.

Validation and Examples

Axi-symmetric flow

The residence-time algorithm is first tested on flow through a converging-diverging duct. This case is selected because of the unusual nature of the solver. This axi-symmetric (with swirl) system was developed by Darmofal[15] and computes the streamfunction as part of the state-vector. This allows the simple calculation of the zero streamfunction or separation surface. All residence time calculations are done in full 3D by spinning the 2D geometry and producing 16 azimuthal sections. The cells are all hexahedra except those at the centerline that are represented as prisms. The following examples were computed with a Reynolds number of 200 and a swirl ratio of 1.75. The duct walls are treated with a slip condition. The solver was run in a transient mode but the input conditions produce a stable flow. Residence-time is calculated for both inviscid incompressible and constant viscosity and density cases.

Figure 1 displays the duct and a rake of streamlines generated from the upstream region. It is clear that something interesting is occurring at the converging part of the pipe.

When seeding streamlines from within that region, it can be seen that there is a part of the flow field that traps the paths. The streamline module in **pV3** stops the integration after a fixed number of integration points to avoid endless compute. Two streamlines are seeded and depicted in Figure 2.

The separation bubble is fully displayed just downstream of the converging section as shown in Figure 3. This is an iso-surface of the streamfunction with value 0.

The characteristic time for this problem is about 24. This is found by looking at an iso-surface of residence-time and finding the time where the major portion of the flow exits the domain. A value greater than that needs to be selected to differentiate old fluid from the core flow. Figure 4 displays an iso-surface of residence-time at the value of 42 when running using the inviscid incompressible formulation.

The tail seen behind the bubble is due to the fact that the flow behind this object is slow. An axi-symmetric comparison of contours can be seen in Figure5.

Figure 6 shows the residence-time iso-surface for the constant viscosity and density case. It should be noted that the tail trailing the bubble is larger indicating the greater mixing due to the viscous stresses.

Flow past a tapered cylinder

Jespersen and Levit[16] computed the viscous incompressible flow about a tapered cylinder. This computation features unsteady vortex shedding at a Reynolds number of 150 based on the mid-span cylinder radius. The mesh used is wraps around the cylinder producing a radial structure that does not quite close upstream. This caused some problems in applying the inflow condition. The call-back `FXmodifyRT` was used to specify the boundary conditions along the upstream open "V" and along the outer "C" portion of the mesh forward of the cylinder. The condition was enforced by specifying $\mathcal{T}_{inflow}$ as X/U_{in}.

The results from the simulation are a series of snapshots in time saved to disk. The time-step between these saved frames is too large for the series to be used for post-processing with this explicit scheme – not every iteration is available. One frame was selected and run as a steady-state problem. The inviscid incompressible residence-time formulation was used - the viscous stresses were not available. The results therefore are not entirely physical but can provide some insight and are displayed in Figure 7.

The base (with contour lines of $\mathcal{T}$) and the tapered cylinder are rendered in gray in Figure 7. Neither the base of the top of the simulation are treated with a no-slip condition. The contour lines are of equal spacing in $\mathcal{T}$ and are roughly parallel (perpendicular to the flow direction) upstream of the post. This indicates undisturbed flow. The effect of the vortices in the wake of the post is obvious when examining the contours of the shadow region. These higher contour values are a reflection of the slower moving flow and the fluid trapped in the (unmoving) vortices. An iso-surface is seen at about the characteristic time for the problem. This surface clearly depicts the wake deficit

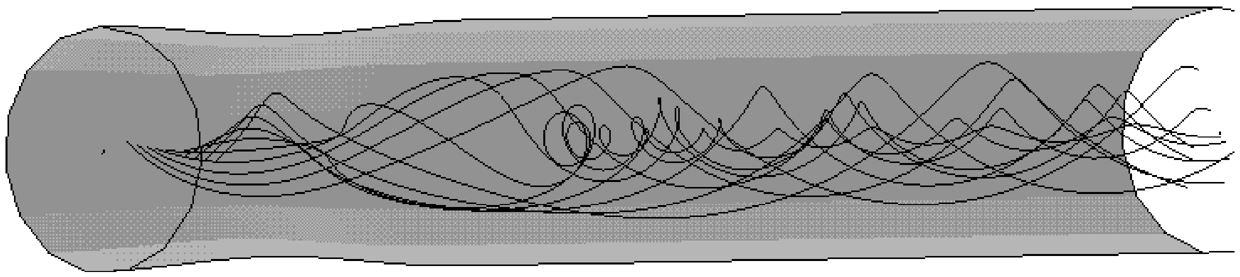

Fig. 1 The geometry with a rake of 10 streamlines seeded from an up-stream position

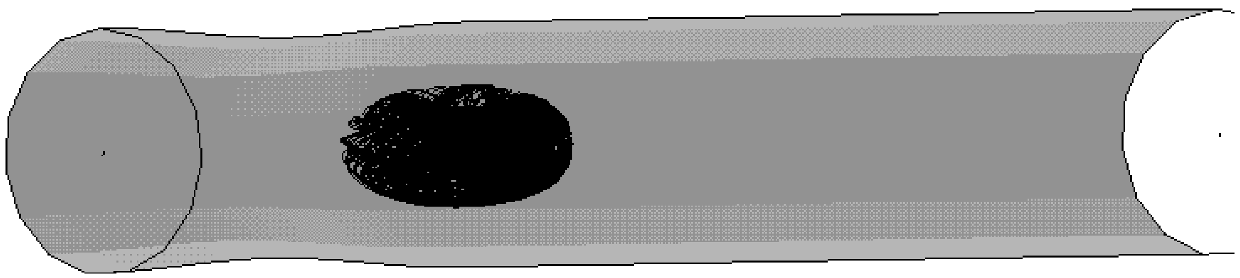

Fig. 2 2 streamlines seeded from within the recirculation region

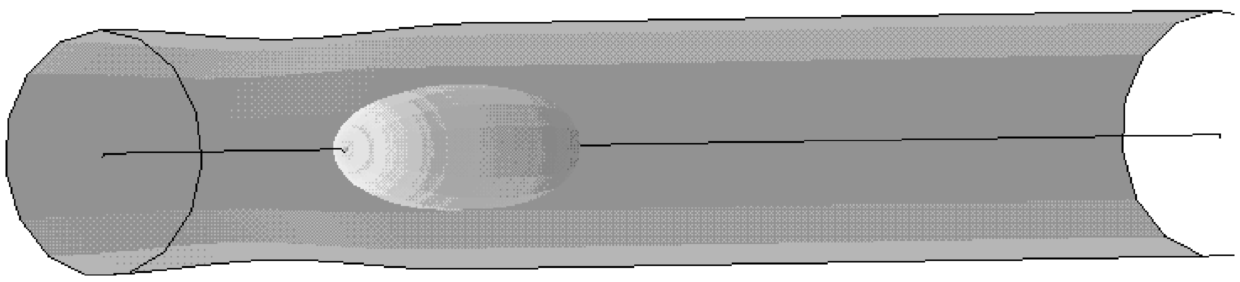

Fig. 3 Streamfunction zero from the solver

downstream and can also be seen attached to the backside of the cylinder where the flow has been trapped.

Conclusions

The residence-time equations for different types of flow conditions (inviscid incompressible, viscous incompressible, inviscid compressible, and constant density/viscosity) have been formulated. An explicit time-marching algorithm of Lax-Wendroff type is used to solve the equations either coupled to a solver or in a post-processing mode. This explicit algorithm performs the computation on a cell-by-cell/node-by-node manner, and thus can be used within the context of **pV3**'s distributed processing. In order to handle the variety of possible boundary-condition treatments, provisions are made to allow the programmer to supply a subroutine where some integration variables can be modified before the residence-time values are updated.

This procedure is inefficient for steady-state post-processing and can not be used for co-processing if the solver formulation is implicit. An accelerated solving procedure is required that can assist in these setting. This procedure must be immune to the fact that there will likely be flow reversal with in the domain of interest.

Acknowledgments

David Sujudi performed much of the initial work on this technique when he was a student at MIT. Mike Giles of Oxford suggested its path. Dave Darmofal was helpful in fixing the numerical smoothing and keeping me honest.

This work was partially sponsored by NAS at NASA Ames Research Center with David Kao as the Technical Monitor and by Army Research Labs with Stephen Davis as the Technical Monitor. Additional support came from the IBM SUR Project and the IBM UUP Project.

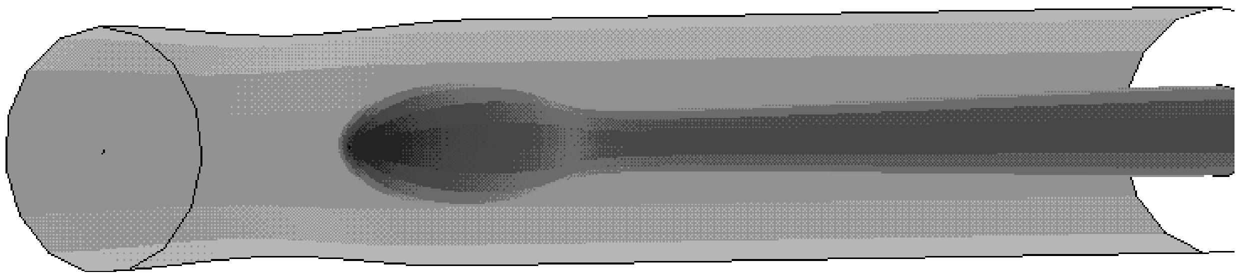

Fig. 4 Iso-surface of residence-time using the invicid formulation

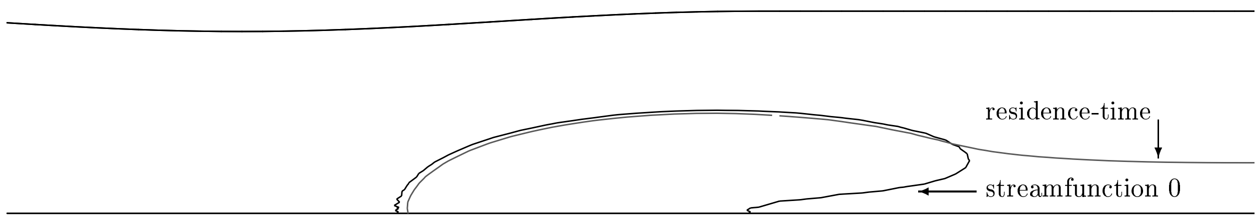

Fig. 5 A comparison between streamfunction 0 and residence-time contours in an axi-symmetric cut

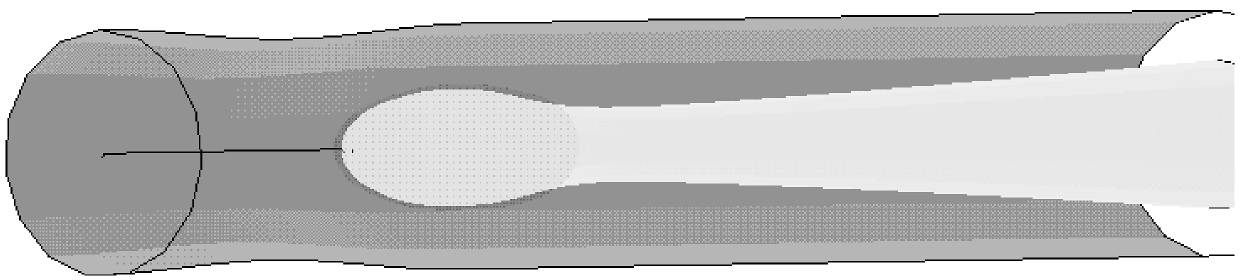

Fig. 6 Iso-surface of residence-time for the constant viscosity and density case

References

[1] J. Helman and L. Hesselink. Analysis and Representation of Complex Structures in Separated Flows. SPIE Proceedings, Vol 1459, 1991.

[2] J. Helman and L. Hesselink. Visualizing Vector Field Topology in Fluid Flows. IEEE Computer Graphics and Applications, May 1991.

[3] David Sujudi. Distributed Visualization and Feature Identification for 3D Steady and Transient Flow Fields. MIT Thesis, 1996.

[4] D. Sujudi and R. Haimes. Identification of Swirling Flow in 3-D Vector Fields. AIAA Paper 95-1715, 1995.

[5] D. Kenwright. Automatic Detection of Open and Closed Seperation and Attachment Lines. Proceedings of IEEE Visualization '98, 1998.

[6] D. Kenwright and R. Haimes Vortex Indetification – Applications in Aerodynamics. Proceedings of IEEE Visualization '97, 1997.

[7] R. Haimes. **pV3**: A Distributed System for Large-Scale Unsteady Visualization. AIAA Paper 94-0321, 1994.

[8] R. Haimes and D. Edwards. Visualization in a Parallel Processing Environment. AIAA Paper 97-0348, 1997.

[9] M. B. Giles, personal communication.

[10] A. Saxer. A Numerical Analysis of 3-D Inviscid Stator/Rotor Interactions Using Non-Reflecting Boundary Conditions. MIT Thesis, 1992.

[11] R.-H. Ni. A Multiple Grid Scheme for Solving the Euler Equations. AIAA Journal, Vol 20, pp. 1565-1571, 1981.

[12] M. G. Hall. Cell-Vertex Multigrid Schemes for Solution of the Euler Equations. Technical Report 2029, Royal Aircraft Establishment, 1985.

[13] R.-H. Ni and J. Bogoian. Prediction of 3-D Multi-Stage Turbine Flow Field Using a Multiple-Grid Euler Solver. AIAA Paper 89-0203, 1989.

[14] D. G. Holmes and S. Connell. Solution of the 2-D Navier-Stokes Equations on Unstructured Adaptive Grids. AIAA Paper 89-1932, 1989.

[15] D. Darmofal. A Study of the Mechanisms of Axisymmetric Vortex Breakdown. MIT Thesis, 1993.

[16] D. Jespersen and C. Levit. Numerical Simulation of Flow Past a Tapered Cylinder. AIAA Paper 91-0751, 1991.

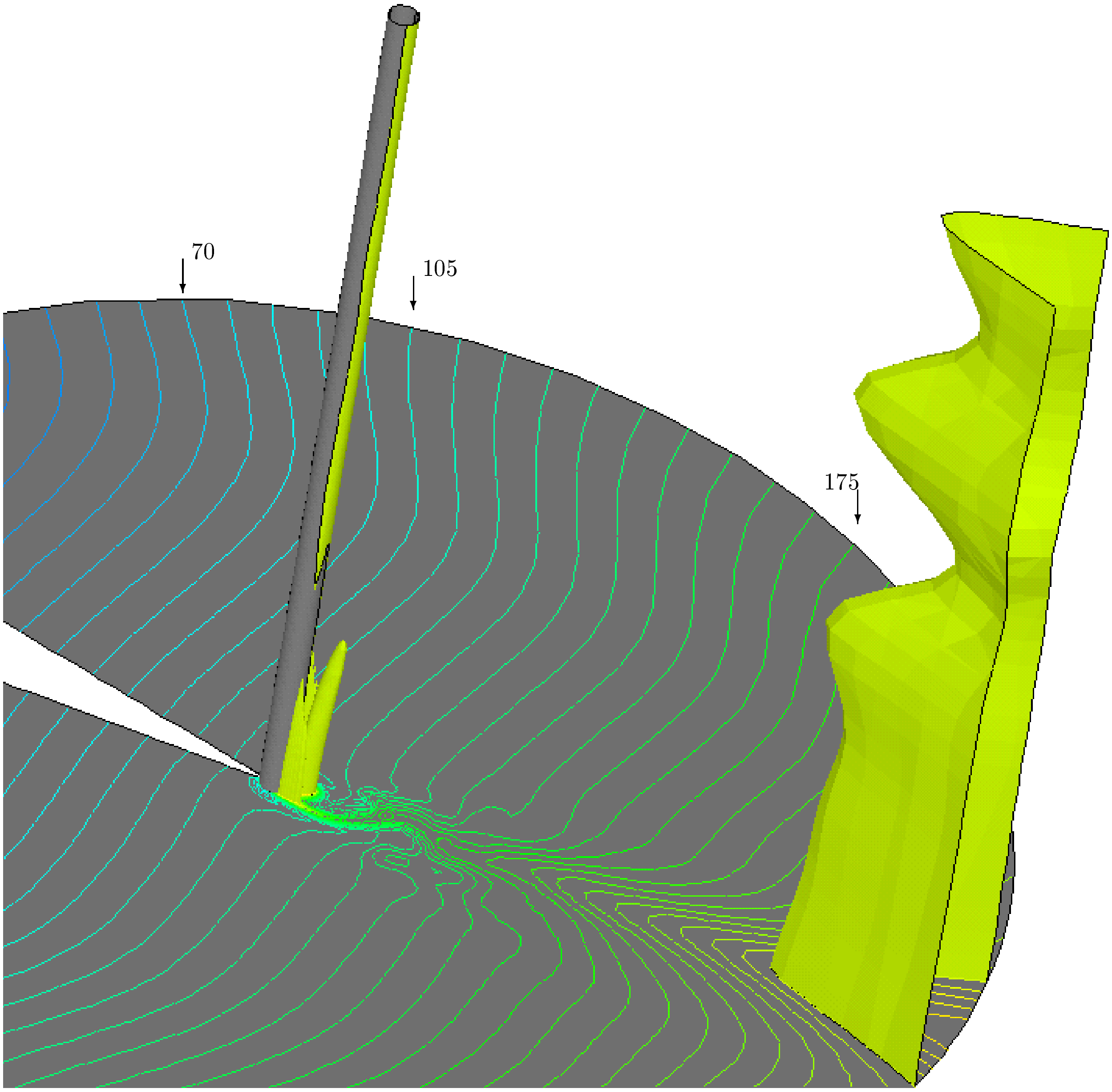

Fig. 7 Tapered cylinder example showing the base with contours and an iso-surface of $\mathcal{T} = 245$.