

# Distributed Visualization and Feature Identification for 3-D Steady and Transient Flow Fields

by

David D. Sujudi

B.S., Aeronautical and Astronautical Engineering  
The Ohio State University, 1991

S.M., Aeronautics and Astronautics  
Massachusetts Institute of Technology, 1994

SUBMITTED TO THE DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS  
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

ENGINEER IN AERONAUTICS AND ASTRONAUTICS

AT THE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

FEBRUARY 1996

© 1996 Massachusetts Institute of Technology. All rights reserved.

The author hereby grants to MIT permission to reproduce and to distribute publicly paper and electronic copies of this thesis document in whole or in part.

Signature of Author: \_\_\_\_\_  
Department of Aeronautics and Astronautics  
January 12, 1996

Certified by: \_\_\_\_\_  
Robert Haimes  
Principal Research Engineer  
Thesis Supervisor

Accepted by: \_\_\_\_\_  
MASSACHUSETTS INSTITUTE OF TECHNOLOGY  
Professor Harold Y. Wachman  
Chairman, Departmental Graduate Committee

FEB 21 1996 Aero

LIBRARIES

# Distributed Visualization and Feature Identification for 3-D Steady and Transient Flow Fields

by

David D. Sujudi

Submitted to the Department of Aeronautical and Astronautical Engineering on January 16, 1996 in Partial Fulfillment of the Requirements for the Degree of Engineer in Aeronautical and Astronautical Engineering

## Abstract

This thesis presents the development of three different visualization tools designed to facilitate the understanding of 3-D steady and transient discretized flow fields in a distributed-computing environment.

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 engines. The decomposition brings up concerns regarding where and how a particle-path or streamline integration can continue when the integration reaches a sub-domain boundary. This thesis presents a scheme which manages the flow of information between the clients, where the integrations are done, and the server, which does the rendering, in order to minimize network traffic, avoid multiple instances of the same object, and make efficient use of the parallel environment.

An algorithm to automatically locate the center of swirling flow has also been developed. The scheme is based on critical-point theory. By using only tetrahedral cells (and transforming other cell types to tetrahedra), the scheme can perform the computation on a cell-by-cell manner and exploit the considerable simplification from using linear interpolation on tetrahedral cells. As such, it can readily take advantage of the compute power provided by a distributed environment.

Finally, a residence-time integration tool, which computes the amount of time the fluid has been in (or in residence within) the computational domain, is developed. The tool is then used to identify separation regions. The fluid in a separated region usually remains there for a considerable amount of time. There will be a significant difference between the residence time of the fluid within the separated flow and that of the surrounding fluid. This difference is used to distinguish the separation region. The residence-time equation for different types of flows (inviscid incompressible, viscous incompressible, inviscid compressible, and constant density/viscosity) are formulated. An explicit time-marching algorithm of Lax-Wendroff type is used to solve the residence time equation on a cell-by-cell manner.

Thesis Supervisor: Robert Haines

Title: Principal Research Engineer

## **Acknowledgments**

I would like to thank Robert Haimes for his support and guidance throughout the course of this work and for the opportunity to work with him on pV3.

To my parents and grandparents, thanks for the support, love, and encouragement. And also for Rini, thank you for the love, support, and patience.

Finally, I would like to thank God for making it all possible.

# Table of Contents

<b>Abstract</b> .....	<b>2</b>
<b>Acknowledgments</b> .....	<b>3</b>
<b>Table of Contents</b> .....	<b>4</b>
<b>List of Figures</b> .....	<b>6</b>
<b>List of Tables</b> .....	<b>10</b>
<b>Nomenclature</b> .....	<b>11</b>
<b>1. Introduction</b> .....	<b>12</b>
1.1 Thesis Outline.....	14
1.2 pV3.....	15
<b>2. Integration of Particle Path and Streamline across Internal Boundaries</b> .....	<b>18</b>
2.1 Particle-Path and Streamline Integration .....	18
2.2 Solution.....	21
<b>3. Swirl Flow Finder</b> .....	<b>28</b>
3.1 Theory.....	30
3.2 Implementation.....	31
3.3 Testing .....	34
<b>5. Residence Time</b> .....	<b>42</b>
4.1 Residence time equation .....	43
4.2 Lax-Wendroff algorithm .....	46
4.3 Boundary conditions.....	49
4.4 Numerical smoothing.....	50
4.5 Sample results .....	52
<b>5. Conclusion</b> .....	<b>66</b>
5.1 Summary of Results.....	66
5.2 Suggestions for Future Work.....	68
<b>Bibliography</b> .....	<b>69</b>

**Appendix A: Formulation of Lax-Wendroff Algorithm .....73**

**Appendix B: Identification of Flow Separation Using the Eigensystem of the  
Velocity-Gradient Tensor .....77**

# List of Figures

Figure 1.1 Interaction between the server and clients in a typical 2-client setup.....	16
Figure 1.2 pV3 user interface. ....	17
Figure 2.1 Illustration of a streamline with 5 segments.....	20
Figure 2.2 Illustration for particle integration in a 3-client setup.....	23
Figure 2.3 Integration of a particle requiring a transfer from client 1 to client 2 and a re-transfer from 2 back to 1. ....	25
Figure 2.4 Illustration of a particle requiring 3 transfers. ....	27
Figure 3.1 Flow pattern at a critical point whose velocity-gradient tensor has 1 real and a pair of complex-conjugate eigenvalues. ....	30
Figure 3.2a A hexahedron (or structured-grid cell) divided into 6 tetrahedra. ....	31
Figure 3.2b A prism cell divided into 3 tetrahedra. ....	32
Figure 3.2c A pyramid cell divided into 2 tetrahedra. ....	32
Figure 3.3 A sample result of an artificially-generated test case. ....	35
Figure 3.4b flow past a tapered cylinder. Swirl flow centers and streamlines are shown.....	37
Figure 3.4a Flow past a tapered cylinder. Shown are swirl flow centers found by the algorithm.	37
Figure 3.5 Flow over a F117 fighter. Swirl flow centers and streamlines are shown. Note: The centers are not mirrored.....	38
Figure 3.4c Blow up of figure 4b.....	38
Figure 3.6a Result of FAST vortex core finder on data set 1.....	39
Figure 3.6b Result of pV3 swirl flow finder on data set 1.....	39
Figure 3.6c Result of FAST vortex core finder on data set 2.....	40
Figure 3.6d Result of pV3 swirl flow finder on data set 2.....	40
Figure 3.6f Result of pV3 swirl flow finder on data set 3. ....	41

Figure 3.6e Result of FAST vortex core finder on data set 3. Note that a streamline has also been spawned here to indicate that the twist at the top of the middle curve is actually outside the vortex. .... 41

Figure 4.1 Pseudo-mesh cell P (dashed) surrounding node  $i$  and a cell neighbor A. The corners of P are composed of the center of neighboring cells marked by capital letters..... 46

Figure 4.2 Face and node numbering for hexahedral cell. .... 47

Figure 4.3 Stencil for pseudo-Laplacian. Only cell A is shown. .... 50

Figure 4.4b Residence-time contours for flow shown in Fig. 4.4a ..... 51

Figure 4.4a Artificially-generated flow with constant velocity in the x direction,  $v_x = 1$ ..... 51

Figure 4.5a Swirling flow through a converging-diverging duct [11]. A separation bubble is shown as an iso-surface of streamfunction with value 0. Streamlines are also shown..... 52

Figure 4.5b Contours of residence time for flow through a converging-diverging duct. .... 53

Figure 4.5c Contours of residence time and iso-surface of streamfunction with value 0. .... 53

Figure 4.5d Iso-surface of residence time of value 27.5..... 53

Figure 4.6a Mesh at the hub of the stator/rotor (stator: 40 x 15 x 15, rotor: 40 x 15 x 15). Dashed line segments indicate periodic boundaries. .... 54

Figure 4.6b 3-D illustration of stator/rotor passage. Grey area is the hub. .... 55

Figure 4.7 Nodes at the stator/rotor interface. Solid circles indicate actual nodes. Unfilled circles indicate locations corresponding to actual nodes at the opposite interface..... 56

Figure 4.8 Nodes at non-overlapping region can be mapped to location at the opposite interface by using the periodicity assumption. .... 56

Figure 4.9a Residence-time contours at the hub. Regions where anomalies occur are indicated by dashed circles. .... 58

Figure 4.9b Residence time contours at the tip. Regions with anomalies are similar to the ones shown in Fig. 4.9a..... 60

Figure 4.10 Residence time iso-surface of value 7.7. For clarity, rotor blade surfaces are not shown..... 61

Figure 4.11 Residence time iso-surface of value 11.0..... 61

Figure 4.13a Mach number contours at the hub. Regions with large gradient indicated by dashed circles. .... 62

Figure 4.12 Region B (of Fig. 4.9a) enlarged to show an unrealistically-large gradient of residence time.....	62
Figure 4.13b Mach number contours. Region (at the stator suction surface near the trailing edge) with large gradient. ....	63
Figure 4.13c Mach number contours at the tip. Regions with large gradient indicate by dashed circles. ....	63
Figure 4.13d Density contours. Density non-dimensionalized by the inlet stagnation density. Regions with large gradient indicated by dashed circles. ....	64
Figure 4.13e Density contours. A region with large density gradient (at the stator suction surface near the trailing edge). ....	64
Figure 4.13f Density contours. A region with large density gradient (near the trailing edge of the stator pressure surface). ....	65
Figure A.1 Pseudo-mesh cell P (dashed) surrounding node i and a cell neighbor A. The corners of P are composed of the center of neighboring cells marked by capital letters.....	73
Figure A.2 Face and node numbering for hexahedral cell. ....	74
Figure A.3 Contributions of cell A to the first-order (left), and the second-order (right) flux integration. ....	75
Figure B.1 Flow patterns for selected distributions of eigenvalues of the velocity-gradient tensor. ....	77
Figure B.2a Flow separating at a 3-degree angle from the specified separation plane (dotted line). Velocity is zero and uniform in the z direction. ....	80
Figure B.2b Side view of separating flow and the surface found by the separation-flow-finder algorithm. ....	80
Figure B.2c Separation surface shown in Fig. B.2b. Velocity vector cloud not shown. ....	81
Figure B.3a Flow moving in opposite direction divided by a separation plane (dotted line). Velocity in is zero and uniform in the z direction. ....	81
Figure B.3b Side view of opposing flow and the surface found by the separation-flow-finder algorithm. ....	82
Figure B.3c Separation surface shown in Fig. B.3b. Velocity vector cloud not shown. ....	82
Figure B.4a Swirling flow through a converging-diverging duct [11]. A separation bubble is shown as an iso-surface of streamfunction with value 0. ....	83



Figure B.4b Surfaces found by the separation-flow-finder algorithm. .... 83

Figure B.4c Enlargement of Fig. B.4b..... 84

Figure B.4d Separation surfaces and streamfunction iso-surface overlapped..... 84

# List of Tables

Table 3.1 Size of Test Cases for Swirl-Flow Finder Algorithm ..... 34

# Nomenclature

F, G, H	the quantities in the residence-time equation that are x, y, and z differentiated, respectively
Q	the non-derivative term in the residence-time equation
$S_x, S_y, S_z$	projected areas on xz, yz, and xy plane of a face of a hexahedral cell
U	the quantity in the residence-time equation that is time differentiated
V	cell volume
X	eigenvector
t	time
u,v,w	velocity
w	reduced velocity
x,y,z	spatial coordinates

## Greek

$\lambda$	eigenvalue
$\mu$	viscosity
$\rho$	density
$\tau$	residence time

## Superscripts

n	index of discretized time
-	averaged quantity

## Subscripts

i, j, k	referring to the Cartesian coordinate directions
i	a node in a discrete computational domain

# Chapter 1

## Introduction

Understanding 3-D transient CFD data is difficult. The additional spatial and time dimensions generate considerably more information than 2-D steady calculations. Furthermore, the increasing complexity of the type of flow being simulated introduces numerous flow features investigators will find of interest, such as vortices, separation bubbles, and shock interactions. All of these directly impact the amount of effort needed to study and interpret such data.

To aid investigators in this effort, various visualization software (as well as software/hardware) packages have been developed. While most of these systems share many similar features, new and unique concepts are constantly being developed in order to achieve the ultimate goal, a better understanding of the data. For instance, the Virtual Windtunnel [5] immerses the user within the data by employing a six-degree-of-freedom head-position-sensitive stereo CRT system for viewing. A hand-position-sensitive glove controller is used for manipulating the visualization tools within the virtual flow field. Systems such as AVS [36] [7] and Khoros [33] [32] uses a visual programming environment in an attempt to ease and expedite the programming process. In the data-flow visual programming environment, the user defines the processing to be done on the data graphically, using icons to represent the processors and links between icons to represent the flow of data. At the other end of the spectrum, pV3 [19] [20] employs a more conventional programming (through subroutine calls) and a straight forward and easy to learn user interface. It is designed for distributed computing and co-processing, allowing for scalable compute power for large data sets.

To facilitate the investigation of the data, these packages provide many common visualization tools, which can be categorized according to their capabilities [8] [16]: feature identification, scanning, and probing. Feature identification tools, such as a shock detector, enable the users to find particular features of the flow quickly. Scanning tools, such as cutting surfaces and iso-surfaces, provide a way to incrementally view the domain by specifying either a spatial or a scalar

parameter. Probing tools, such as instantaneous streamlines, streamline probe, and point probe, provide the most localized information, the behavior of the flow or the values of a particular flow quantity at certain point(s). This categorization also reflects the reduction of dimensionality of the information in the data, from 3-D to 0-D, which is sometimes necessary in understanding the various aspects of the flow.

These tools can be effective because they provide the users information based on the data in a form that can be absorbed more easily. For instance, streamlines show how the vector fields behave, and using it is usually more effective in comprehending the flow field than looking at pages of raw numbers. Iso-surfaces provide information on the scalar fields in the data by generating surfaces where the scalar field has a specified value, which concisely describes a particular distribution within the domain.

Also crucial in the effectiveness in the use of these tools is the speed at which they can be manipulated interactively. In investigating the vast amounts of data produced by the CFD solvers, the users will need to orient and re-orient cutting surfaces around the domain, spawn streamlines, and generate iso-surfaces. An almost instant feedback is required so as not to break the users' train of thought during the investigation. For any particular tool, the feedback time depends mainly on the size of the domain, the capability of the computer, and the number of other tools being requested.

In recent years, unsteady 3-D simulations have become more common, and their size has increased steadily. Along with this growth comes a demand for a visualization software capable of handling huge 3-D unsteady data as well as providing tools for interrogating that data. To meet this goal, a parallel visualization software, called pV3 has been developed at MIT [19] [20]. pV3 is designed for co-processing and also distributed computing. 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 engines. Thus, the processing needed by the visualization-tool algorithms (e.g., finding iso-surfaces, integrating particles and streamlines, etc.) can be done in parallel. With parallel processing, the compute power can be scaled with the size of the problem, thus maintaining the desired interactivity.

The computation of some of the visualization tools, such as iso-surfaces and cutting planes, are embarrassingly parallel since they are done on a cell-by-cell basis. As such, these computations are automatically parallelized when the computational domain is decomposed. However, domain decomposition brings up concerns regarding the where and how a particle-path or streamline integration can continue when the integration crosses a sub-domain boundary (also called internal boundary). This is not an issue in a single-domain environment, in which a particle-path or streamline calculation stops when the integration reaches the domain boundary. One of the aims of this thesis is to present a method for managing the information movement needed to continue integrations across these internal boundaries.

Parallel processing also invites opportunities to develop visualization tools that can take advantage of distributed compute power. As noted by Haines [16], the weakest link in visualization is probably the scarcity of feature identification tools. Thus, the second and third thrust of this work is to present two new feature identification tools, a swirling-flow finder and a residence-time integrator, whose computations are designed to be parallel. A swirling-flow finder automatically identifies the center of swirling flows and displays the center as a series of disjoint line segments. A residence-time integrator computes the amount of time the fluid has been within the computational domain. These tools provide alternative ways of looking at the data and its features, and it is hoped that they can present new insights which lead into better understanding of the data and the underlying physics.

## **1.1 Thesis Outline**

Since all the work and development that will be presented here are done within the context of pV3, a brief description of pV3 and its environment will be given in the next section.

Chapter 2 describes how domain decomposition effects the integration of streamlines and particle paths, and the issues and problems encountered in continuing integration across sub-domain boundaries. A solution that takes into consideration issues such as minimizing network traffic and maximizing the use of the parallel environment will then be presented.

Chapter 3 first presents the motivation for the development of the swirling-flow identification tool. The fundamental theory and its implementation will then be explained. And finally, some result validation will be shown.

The algorithm used in computing residence time is discussed in chapter 4. The development of the governing equations of residence time for various types of flow (inviscid incompressible, inviscid compressible, viscous compressible, and constant density and viscosity) and some sample calculations will also be presented.

Finally, the contributions of this work will be summarized in chapter 5.

## 1.2 pV3

pV3 is the latest in a series of visualization software developed at MIT. The previous generation software, Visual2 [13] and Visual3 [17] [18], are designed for visualizing two and three dimensional data in a post-processing manner. On the other hand, pV3 is designed for co-processing visualization of unsteady data whereby the user can visualize the results as the model or solver progresses in time. pV3 also allows the solution to be computed in a distributed/parallel environment, such as in a multi-workstation environment or in a parallel machine. In such a setup, the movement of visualization data between the solver machines and the visualization workstation will be taken care of by pV3. The development of pV3 was motivated by the increase in size of 3-D unsteady calculation and the need for highly interactive tools in visualizing and interrogating the results [19].

In the distributed solver model of pV3, the CFD system/solver decomposes the computational domain into sub-domains. The computational domain is made up of elements (such as tetrahedra, hexahedra, etc.), and the sub-domain boundaries are formed by the facets of these elements. pV3 accepts any combination of these elements: disjointed cells (tetrahedra, pyramids, prisms, and hexahedra), poly-tetrahedral strips, and structured blocks. In pV3, the programmer can specify where a streamline/particle-path integration should continue (to a cell in a specific sub-domain, through a specific internal boundary in a sub-domain, or to try all sub-domains) when the integration reaches a sub-domain boundary.

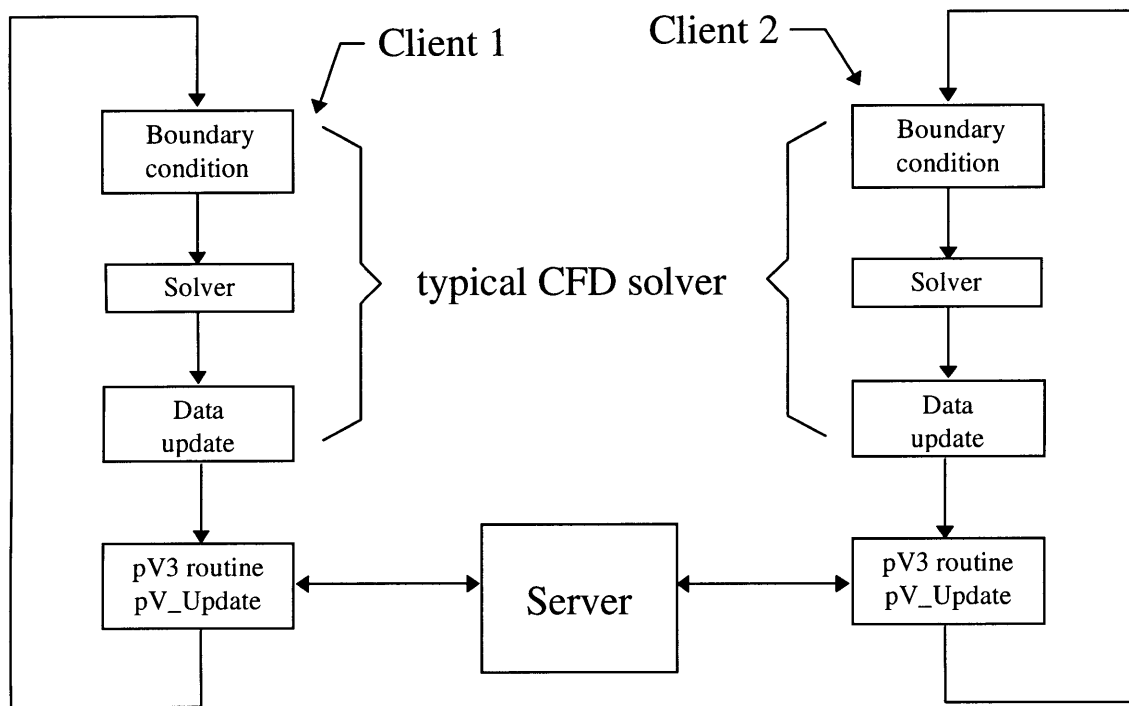


Figure 1.1 Interaction between the server and clients in a typical 2-client setup.

Each sub-domain is handled by a separate process (called a client in pV3), and each computer in the distributed environment can execute one or more client(s). The user interface and graphic rendering are handled by a process (called the server) run on a graphics workstation. As an illustration, the interaction between the server and the clients in a typical 2-client setup is shown in Fig. 1.1. A call to a pV3 subroutine (named pV\_Update) is added to a typical CFD solver to handle the processing needed by pV3, including particle-path and streamline integrations. To keep the processing synchronized, each client does not exit pV\_Update until the server transmits a time-frame-termination message. Only then can the clients continue to the next time step.

From the user's point of view, pV3 consists of a collection of windows, which includes a 3D, 2D, and 1D window, as illustrated in Fig. 1.2. Objects such as computational boundaries, iso-surfaces, streamlines, or cutting planes are shown in the 3D window. The mapping of the cutting plane or contours are shown in the 2D window. The 1D window displays one dimensional data generated by various probing functions of the 2D window or from mapping the values traced along a streamline. This arrangement, in conjunction with reduction of dimensionality provided



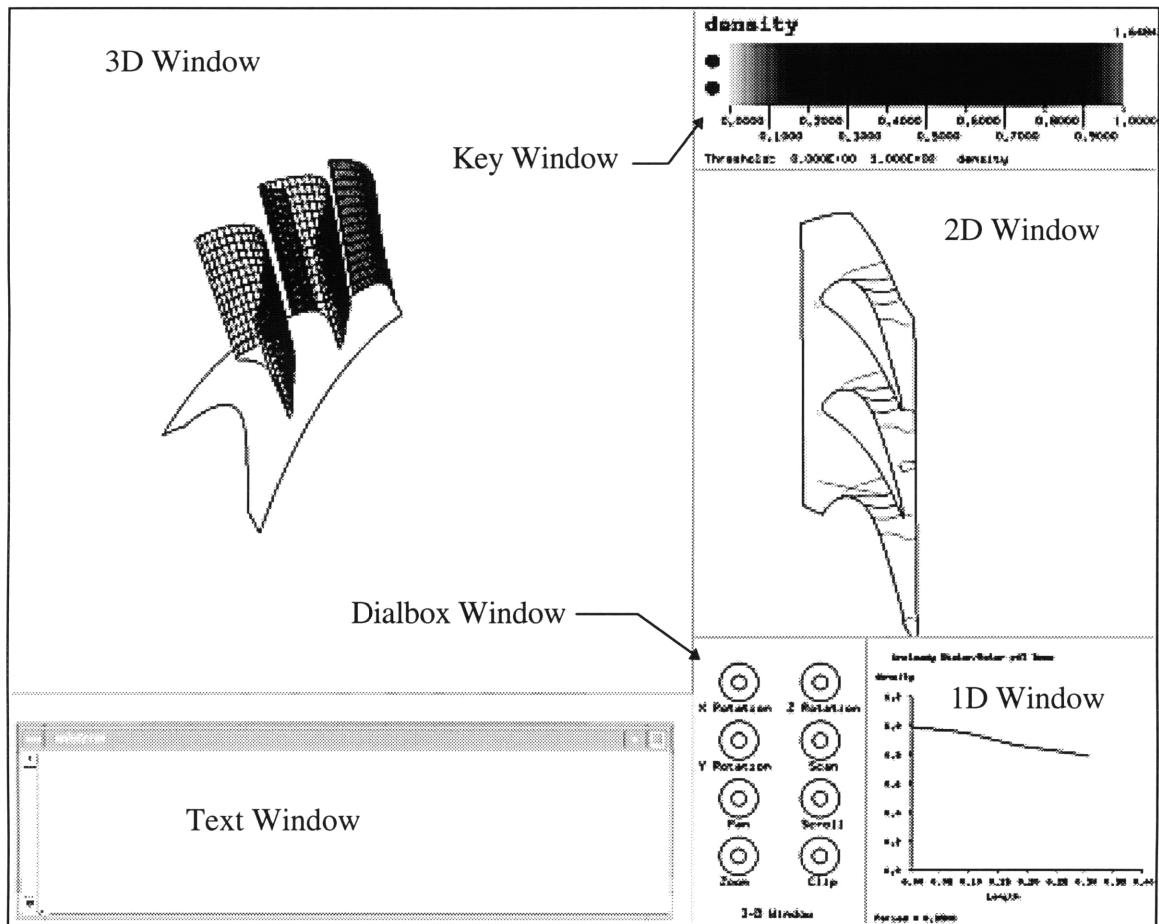


Figure 1.2 pV3 user interface.

by the visualization tools, enable the user to simultaneously consider several aspects of the data, and thus facilitating the comprehension of complex 3-D data.

Interactions with pV3 - for orienting objects in the 2D or 3D window, activating visualization tools, modifying tool parameters, etc. - are done through mouse clicks and keyboard inputs within the various windows. By entering text strings in the pV3 text window, the user can interact with the solver by sending messages which are then interpreted by programmer-supplied routines in the solver. This capability, which is unique among visualization software, enables the user to steer the solution while simultaneously visualizing the results.

## Chapter 2

# Integration of Particle Path and Streamline across Internal Boundaries

### 2.1 Particle-Path and Streamline Integration

A streamline is a curve in space which is everywhere tangent to the instantaneous velocity field. It is calculated by integrating:

$$\frac{d\vec{x}}{d\alpha} = \vec{u}(\vec{x}) \quad (2.1)$$

where  $\vec{x}$  is the position vector,  $\vec{u}$  the velocity vector, and  $\alpha$  the pseudo-time variable. The integration employs a fourth-order Runge Kutta method with variable pseudo-time stepping. The pseudo-time variable is used only for integrating streamlines and is independent of the actual computation time-step.

A particle path, on the other hand, represents the movement of a massless particle as time progresses. The calculation amounts to integrating:

$$\frac{d\vec{x}}{dt} = \vec{u}(\vec{x}, t) \quad (2.2)$$

The integration uses a fourth-order accurate (in time) backward differencing formula. Details about the streamline and particle path integration algorithms can be found in [9] [10].

Both types of integrations start at a user-specified seed point and end when a computational boundary is reached. However, between these points the integration might pass through one or more sub-domain boundaries. When this occurs, information needed in continuing the integration (referred to as a “continue-integration request” or “transfer request”) must be sent to other client(s) so that the integration can continue. The information usually comprises of integration state, identification number for the particle/streamline, rendering options, the identification of the

client sending the request, etc. As stated in section 1.2, the CFD system can specify where an integration should continue when it crosses an internal boundary. However, this specification might be erroneous, or it might not be applicable in certain flow conditions. Thus, for robustness, the system should try its best to continue an integration despite an incorrect instruction. For this reason the client(s) receiving the transfer request has the option to do one of the following:

- accept the transfer: determines that the integration continues in its sub-domain and proceeds with the integration.
- reject the transfer: determines that the integration does not continue in its sub-domain and ignore the request.
- request a re-transfer: determines that the integration does not continue in its sub-domain and requests that the integration be continued in one or more other client(s).

The algorithm for determining whether an integration hits an internal boundary, or whether a client accepts/rejects a transfer, or requests a re-transfer involves checking the spatial location used in the current integration stage against the sub-domain space. The details of this method will not be discussed in this thesis, but can be found in [9]. Here, only the problems encountered in managing transfer requests will be addressed, as well as how the client(s) and the server must interact to ensure that integrations are continued, while keeping the process as efficient as possible. The issues can be stated with the following questions:

- Due to the possibility of re-transfer, how can sending the same request to the same client(s) repeatedly be avoided? This is a question of efficiency as well as avoiding multiple instances of the same object within a sub-domain.
- How can the server know when to safely send the time-frame-termination message? Sending this message when there is a possibility of further integration transfers could prematurely abort the integration of some particles or streamlines. Thus, the server must know when all transfers are complete for the current time step. Otherwise, the time-frame-termination signal can not be sent and the clients (involving the CFD solver) will remain in a wait state, stalling the entire calculation. In this case, the graphics server will eventually time-out due to

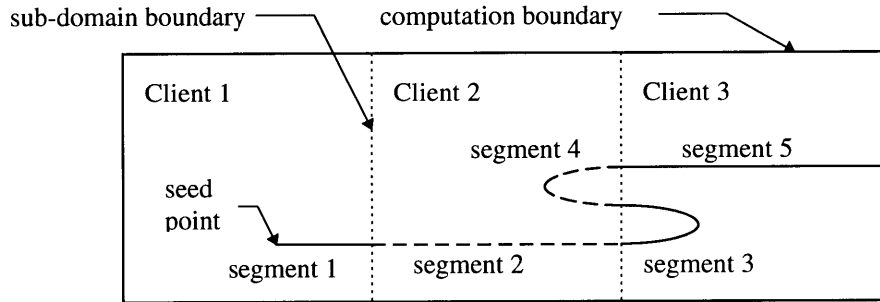


Figure 2.1 Illustration of a streamline with 5 segments

inactivity, release the clients from this wait state, and quit. This situation is clearly undesirable.

- What is the best way to maximize the use of the parallel environment in integrating streamlines and particle paths? These integrations are essentially a serial process. The integration starts at some point at the beginning of the time step and stops at another point at the end of the time step. In between, the process must be done in a certain order. In pV3's parallel environment, the work of integrating streamlines/particles can be distributed because each client processes only the objects in its sub-domain. However, problems in optimally exploiting the distributed environment can occur due to the need to continue integration across internal boundaries and also the existence of other types of requests (most of which are embarrassingly parallel) that the clients need to handle. It can be foreseen that there could be cases where a mixture of transfer requests and other requests could generate a load imbalance between the clients. The problem can be alleviated (thus, maximizing the use of the parallel environment) by prioritizing requests in a certain manner. This issue will be illustrated and made clearer when the solution is discussed in the next section.

When it comes time for rendering, each client sends the server the relevant information about all the particles (i.e., their locations) that end up in its sub-domain at the end of the current time step. A streamline, however, is divided into segments. A new segment begins every time a streamline integration crosses an internal boundary. For example, a streamline with 5 segments is illustrated in Fig. 2.1. To render a streamline, each client sends the server information about the

segments (e.g., the points forming the segment) within its sub-domain. The server combines the segments, constructs a complete streamline and then renders it.

Although, the underlying construction of a particle and a streamline are completely different, the issues encountered in continuing their integration across an internal boundary are in fact similar. For a particle, the question is where it should end up at the end of the current time step. For a streamline, the question is where its next segment should begin at the end of the current pseudo-time step. This similarity will be exploited in developing a scheme that is usable in both cases. The evolution of this scheme will be described in the next section.

## 2.2 Solution

First, the following scheme is proposed:

1. It is decided to process transfer request (TR) through the server instead of having the clients communicate directly. Thus, to continue the integration of a particle/streamline, a client sends a TR to the server, which then distributes it to one or more client(s), depending on the particular internal-boundary specification. This solution provides a simpler and cleaner system overall because all other visualization message traffic is client-server and only one process (the server) needs to manage the transfer requests. Since solver messages are client-client, the risk of interfering with solver messages is also minimized.
2. For each particle/streamline segment that requires transfer across internal boundary, the server keeps a list of clients to which the TR have been sent. The server also assigns a unique identification number (denoted by TID, for “transfer id”) for each of these particles/streamline segments.
3. To prevent sending the same TR to a client, the server can send a TR for a TID to a particular client only if the client is not already in the client list. Thus, to avoid sending a transfer request back to the originating client (i.e., the client which initially sends the transfer request), the originating client must automatically be logged in the client list.

4. Transfer requests have higher priorities (at the client and the server) over other types of visualization requests. Thus, if the request queue of a client or the server contains a TR, that request will be handled before other types of requests. In most situations, this procedure will help lessen load imbalances. As an example, consider a 3-client setup where the request queue of client 2 contains  $R_1$ ,  $R_2$ , and lastly a transfer request TR, while that of client 1 and 3 contains only  $R_1$  and  $R_2$ .  $R_1$  and  $R_2$  could be requests to calculate iso-surfaces, find swirling flow, generate cut planes, etc. Let's suppose that TR will generate a re-transfer to client 3. Requests  $R_1$  and  $R_2$  could take much longer to process in client 2 than in client 3, depending on the size of the sub-domains, the type of request, and the flow conditions. If the clients handle the requests according to their order in the queue (i.e.,  $R_1$  first,  $R_2$  second, and so on), client 3 will finish before client 2. Client 3 will have to wait until client 2 handles the transfer request before it can process the re-transfer that will be generated by that request. On the other hand, if TR has higher priority and is processed first by client 2, client 3 will receive the re-transfer sooner. The entire process will take less wall-clock time than without prioritizing.
5. After a client processes a TR - whether accepting, rejecting, or re-transferring the request - it sends a transfer-processed acknowledgment (TPA) back to the server. Since the originating client is automatically logged in the client list, a TPA is also automatically associated with the originating client.
6. The transfer process for a particular TID is completed when the server receives TPAs from all the clients to which the TID has been transferred. To simplify the algorithm for checking transfer completion it must be ensured that when a client sends a TR and then a TPA (such as in a re-transfer request), the server receives them in the same order. Then, the server will record the re-transfer (if any is allowed) before the TPA. This can be accomplished by setting TPA's priority to be the same as TR's. Now, determining transfer completion can be done at any moment by identifying those TIDs whose client list has a complete set of TPAs. When all TIDs have a matching set of clients and TPAs, the server knows that no more transfers will be requested during the current time step.

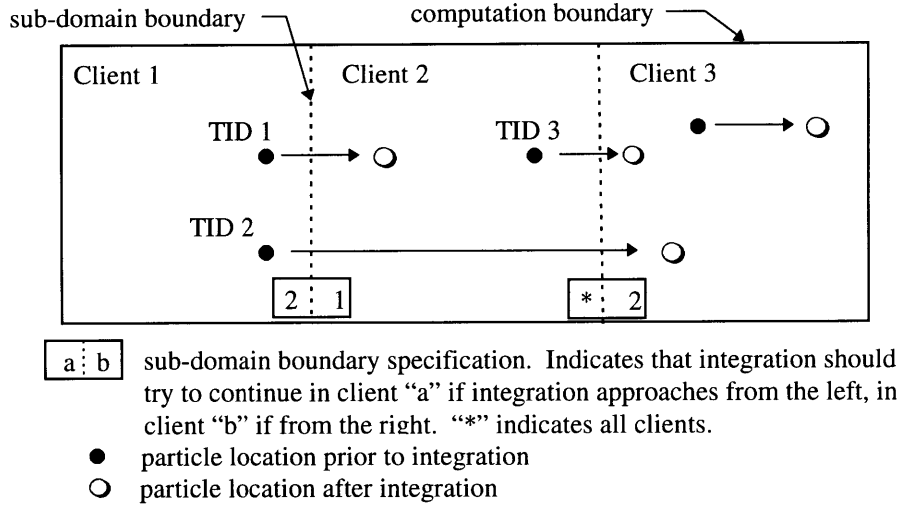
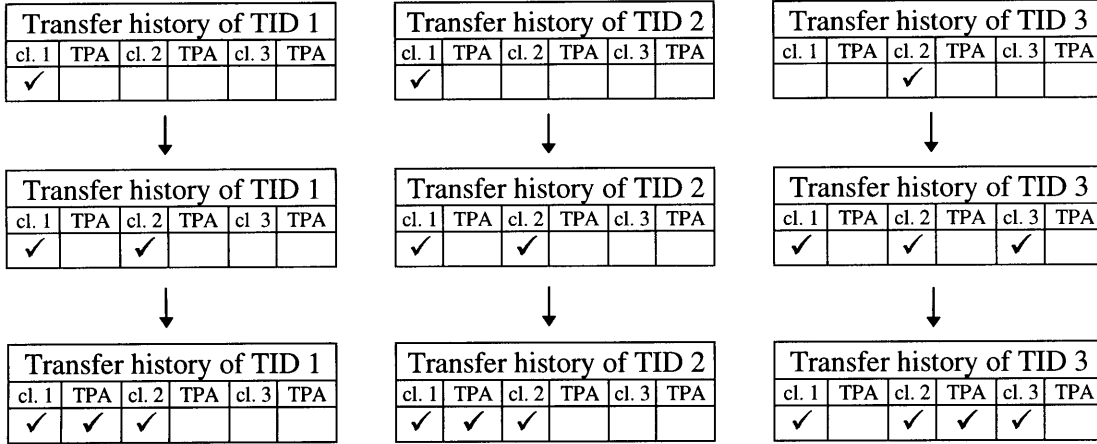


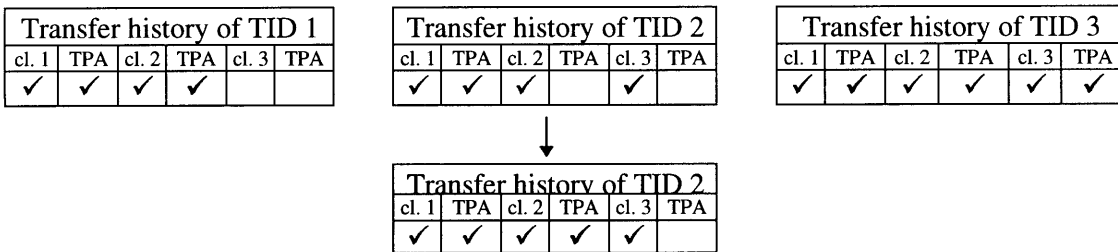
Figure 2.2 Illustration for particle integration in a 3-client setup.

To illustrate this scheme, a 3-client example is shown in Fig. 2.2. In this example 3 particles require transfers in the current time-step and 1 particle does not. Note that no transfer will be requested for the fourth particle (the one in client 3) because it stays within client number 3. Due to the similarity in continuing particle and streamline integration, only examples for particles will be shown. In a step-by-step manner, the scheme works as follows (at each step, the particle transfer log for each TID is summarized in the corresponding table):

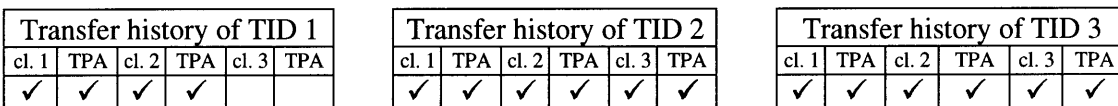
1. The server receives two transfer requests from client 1 and one from client 2, and assigns them TID 1, 2, and 3 as shown in Fig. 2.2. In the tables below, "cl." stands for client, and a "✓" under a client number indicates that a TR has been sent to that client (or the client is the originating client), and a "✓" under TPA means that the server has received a TPA from the corresponding client. Complying with the internal-boundary specification, the server transfers TID 1 and 2 to client 2, and attempts to transfer TID 3 to client 1, 2, and 3. However, since TID 3 already has client 2 in its transfer list, TID 3 will not be transferred there again. Note that a TPA is automatically assigned to the originating client.



2. Client 1 will reject TID 3 and send a TPA. Client 2 will accept TID 1 and send a TPA. Client 2 will also request that TID 2 be re-transferred to all other clients, and then send a TPA for TID 2. However, since client 1 and 2 are already in the client list of TID 2, TID 2 will only be transferred to client 3. Client 3 will accept TID 3 and send a TPA. At the end of this step the transfer process for TID 1 and 3 is complete because each client in their list is paired with a corresponding TPA.



3. Client 3 accepts TID 2 and sends a TPA. At this point (and only at this point), all TID have a complete list of TPAs (i.e., each client in the list is paired with a TPA). Thus, all transfers are complete for this time step.





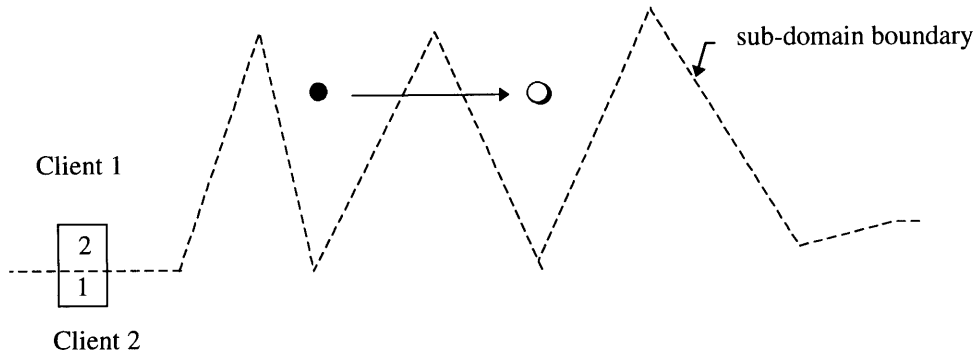


Figure 2.3 Integration of a particle requiring a transfer from client 1 to client 2 and a re-transfer from 2 back to 1.

Now consider the situation illustrated in Fig. 2.3. Initially, client 1 transfers the particle to client 2, then client 2 requests a re-transfer back to client 1. However, since the particle originally comes from client 1, the above scheme will not allow this, and the integration stops. To take care of cases such as this, the above scheme is modified to allow a particle (or streamline segment) to be transferred to the same client twice. Since multiple transfers to the same client are now allowed, there is no need to automatically include the originating client in the client list. For the situation in Fig. 2.3, the process goes as follows:

1. The server receives a transfer request from client 1, and transfer the integration to client 2. In the table below, there are now 2 columns under client and TPA to reflect that the particle can be transferred to the same client twice.

Transfer history of TID 1							
Client 1		TPA		Client 2		TPA	
				✓			

2. Client 2 requests that the particle be re-transferred to client 1, and then sends a TPA.

Transfer history of TID 1							
Client 1		TPA		Client 2		TPA	
✓				✓		✓	

3. Client 1 accepts the transfer and sends a TPA. At this point every check mark in the client column is paired with a check mark in the TPA column, indicating the transfer process for the current time step is complete.

Transfer history of TID 1							
Client 1		TPA		Client 2		TPA	
✓		✓		✓		✓	

This solution, however, is not without trade-offs and limitation. In cases where re-transfers to the same client are not needed (such as in the first example above), unnecessary multiple transfers to a client might be made. In some situations, a client might accept a transfer twice, creating 2 instances of the same particle (or streamline segment). To prevent this, every time a client receives a transfer request it checks whether it has previously accepted the object. If it has, the request is ignored. The checking is done by comparing the global identification number, which is unique through the duration of the visualization session, or the TID, which is unique during each time step.

This solution is also not a general one. Consider the situation shown in Fig. 2.4, in which the particle needs 3 transfers to client 2. This will not be possible without increasing the maximum number of transfers to 3, which will further reduce the efficiency of the scheme. It can be seen that however large this number is set to, the solution will never be general. Thus, as a trade-off between efficiency and more generality the number of transfers have been limited to 2. This limit will cover most cases without significantly compromising efficiency. Should a particle-path integration require more than two visits to the same client, the integration will abort and the particle will be “lost”. If it is a streamline integration, only part of the streamline will be rendered (from the seed point to the last point before the unsuccessful integration transfer). Fortunately, this limit is not too severe in most instances. The streamline integration algorithm uses a pseudo-time step limiter, which is based on the cell size as well as local vector field data. This limiter insures that the next requested point in the integration is no more than the cell’s size from the

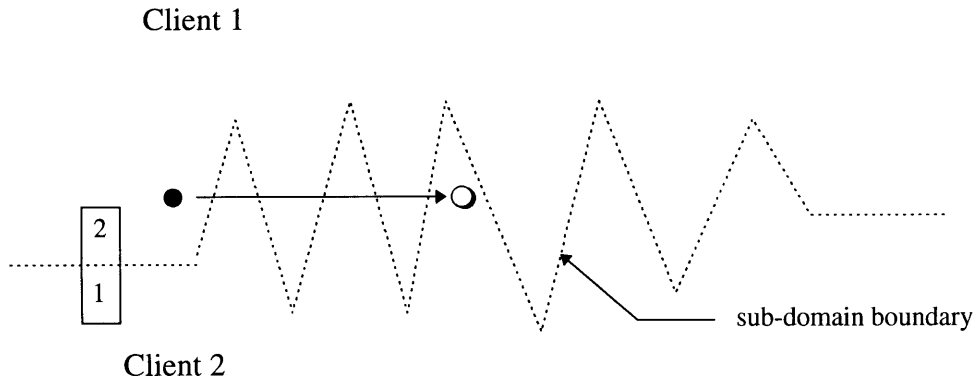


Figure 2.4 Illustration of a particle requiring 3 transfers.

current position. If the size of the neighboring elements do not change drastically, the problems posed by this scheme will be minimal. In a flow computed using an explicit scheme, the time step governed by the CFL stability requirement of the flow solver will generally limit the movement of a particle to no further than the adjacent cells.

## Chapter 3

# Swirl Flow Finder

This work is motivated by the need to easily locate vortices in large 3-D transient problems. A tool that will automatically identify such structures is definitely needed to avoid the time-consuming and tedious task of manually examining the data. However, the question of what defines a vortex raises considerable confusion. As a result, various definitions of a vortex and methods for identifying vortices have been proposed by investigators.

Moin and Kim [28] [27] propose the identification of vorticity using vorticity lines, which are integral curves of vorticity. However, this method is found to be very sensitive to the starting point of the integration. Moin and Kim [28] indicate that a badly chosen initial point will likely result in a vortex line which wanders over the entire flow field, making it difficult to identify any coherent structures.

Chong, Perry, and Cantwell [6] propose a definition based on the eigenvalues of the velocity-gradient tensor. A vortex core is defined as a region with complex eigenvalues, which means the local streamline has a helical pattern when viewed in a reference frame moving with the local flow.

By incorporating some of the work of Yates and Chapman [39] and Perry and Hornung [31], Globus, Levit, and Lasinski [14] identify vortices by first finding a velocity critical point where the velocity-gradient tensor has complex eigenvalues. The vortex core is found by integrating, starting from the critical point, in the direction of the eigenvector corresponding to the only real eigenvalue.

Banks and Singer [3] [4] propose an algorithm which finds the vortex core by using a predictor-corrector scheme. The vorticity vector fields is used as the predictor and the pressure gradient as the corrector. This scheme is designed to self-correct toward the vortex core.

Jeong and Hussain [25] propose a definition of a vortex in incompressible flow in terms of the eigenvalues of the tensor  $S^2 + \Omega^2$ , where  $S$  and  $\Omega$  are the symmetric and antisymmetric parts

of the velocity gradient tensor. A vortex core is defined as a connected region with two negative values of  $S^2 + \Omega^2$ .

These are merely a sample of the works that have been done on this subject. For a more thorough survey, the reader can refer to [4] and [25], both of which discuss the inadequacies and limitations of the various schemes.

Considering the confusion that still exists on what constitutes a vortex and the limitations of existing schemes, this author believes that there is a need for a tool that can help investigators locate vortices, and yet has a familiar and intuitive interpretation. To achieve this goal, the condition that what the tool finds must be vortices will be relaxed. However, the scheme must be practical, in terms of computational speed and usability.

This work proposes that a tool which identifies the center of swirling flows satisfies the above need and criteria. Investigators have been using swirling flow as one of the means to locate vortices in 3-D discretized vector fields. Swirling flows in a 3-D field are usually identified by studying vector fields which are mapped onto planar cuts or by seeding streamlines. These procedures can be very laborious, especially for large and complex flows. The scheme presented here allows automatic identification of the center of swirling flows in 3-D vector fields.

The algorithm for implementing this tool is based on critical-point theory. As will be described below, the scheme works on a cell by cell basis, lending itself to parallel processing, and is flexible enough to work with the various types of grids supported by pV3. With this method, the need for curve integration (which is needed for visualization tools such as streamlines and particle paths) has been avoided. Curve integration is a serial operation and can not readily take advantage of pV3's distributed computing capabilities. And, as described in chapter 2, integrations across a distributed environment involve passing information between machines and other additional complexities which further reduce efficiency.

In the next sections, the theoretical background of this algorithm and its implementation will be described. The results of the scheme on exact artificially-generated data as well as CFD data will be shown. Results will also be compared (using artificially-generated data) against those from a tool developed by Globus, Levit, and Lasinski [14].

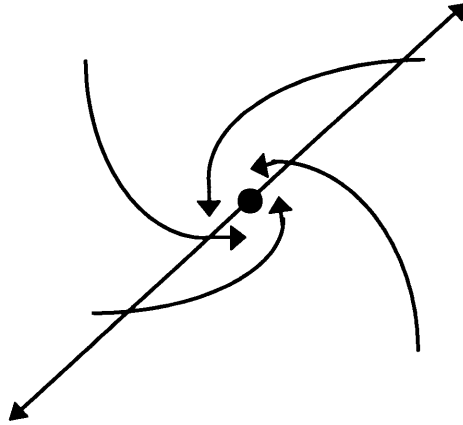


Figure 3.1 Flow pattern at a critical point whose velocity-gradient tensor has 1 real and a pair of complex-conjugate eigenvalues.

### 3.1 Theory

Critical points are defined as points where the streamline slope is indeterminate and the velocity is zero relative to an appropriate observer [6]. According to critical point theory, the eigenvalues and eigenvectors of the velocity-gradient tensor,  $\partial u_i / \partial x_j$  (this matrix will be called  $A$ ), evaluated at a critical point defines the flow pattern about that point. Specifically, if  $A$  has one real and a pair of complex-conjugate eigenvalues the flow forms a spiral-saddle pattern, as illustrated in Fig. 3.1. The eigenvector corresponding to the real eigenvalue points in the direction about which the flow spirals, and consequently, the plane normal to this eigenvector defines the plane on which the flow spirals. For a complete description of all other possible trajectories the reader can refer to [6] or [1].

The pattern in Fig. 3.1 is intuitively recognized as swirling flow, and, therefore the above method can be used to find the center of swirling flows located at critical points. However, there are obviously swirling flows whose center is not at a critical point. Fortunately, a similar method can be applied in these cases.

At a non-critical point with the necessary eigenvalue combination (i.e., one real and a pair of complex conjugates) the velocity in the direction of the eigenvector corresponding to the real eigenvalue is subtracted. The invariance of the eigenvectors' directions with respect to a Galilean transformation ensures that the resulting flow will have the same principal directions. The resulting velocity vector will be called the reduced velocity. If the reduced velocity is zero, then the point must be at the center of the swirling flow. A similar statement was also made by Vollmers, Kreplin, and Meier [38].

Therefore, to find a point at the center of a local swirling flow, the algorithm searches for a point whose velocity-gradient tensor has one real and a pair of complex-conjugate eigenvalues and whose reduced velocity is zero.

### 3.2 Implementation

pV3 accepts structured and/or unstructured grids (containing any combination of tetrahedra, polytetrahedra strips, hexahedra, pyramids, and prism cells). In the interest of efficiency, only tetrahedral cells are used, with all other cell types reduced to 2 or more tetrahedra. Fig. 3.2 shows how various types of cells are decomposed into tetrahedral cells.

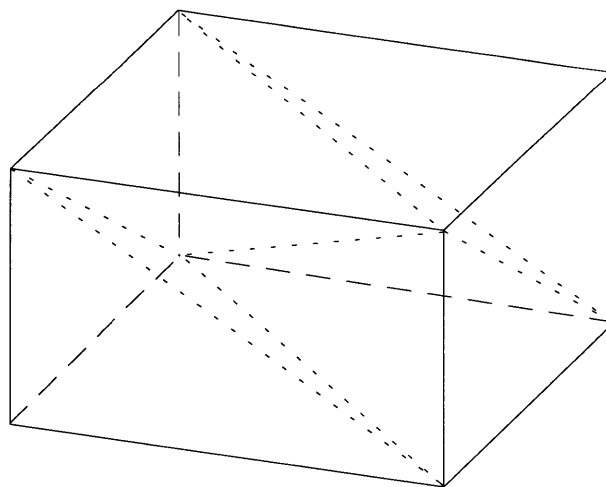


Figure 3.2a A hexahedron (or structured-grid cell) divided into 6 tetrahedra.

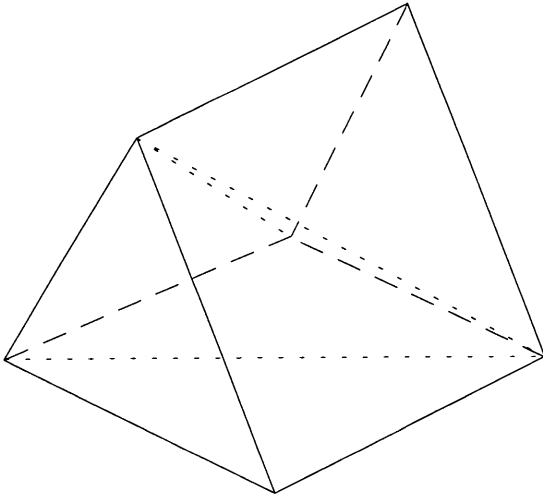


Figure 3.2b A prism cell divided into 3 tetrahedra.

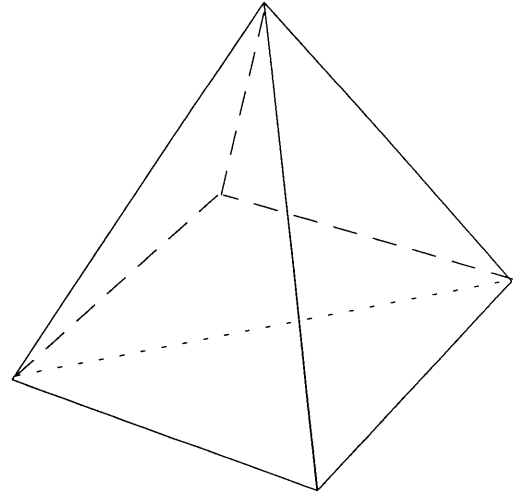


Figure 3.2c A pyramid cell divided into 2 tetrahedra.

This approach allows the use a simple linear interpolation for the velocity, avoiding the more complex, and inherently more costly, interpolation required by other types of cells (such as bilinear interpolation for hexahedra). A tetrahedron has 4 node points, sufficient to solve for the four coefficients of a 3-D linear interpolant. More importantly, linear velocity interpolation produces a constant velocity-gradient tensor within the entire tetrahedral cell. Consequently, the straight forward algorithm described below can be employed, which otherwise would not have been possible.

The algorithm proceeds one tetrahedral cell at a time, and can be summarized as follows (it is assumed that a velocity vector is available at each node):

1. Linearly interpolate the velocity within the cell.
2. Compute the velocity-gradient tensor A. Since a linear interpolation of the velocity within the cell can be written as

$$u_i = C_i + \frac{\partial u_i}{\partial x} \Delta x + \frac{\partial u_i}{\partial y} \Delta y + \frac{\partial u_i}{\partial z} \Delta z \quad (3.1)$$

then A can be constructed from the coefficients of the linear interpolation function of the velocity vector.



3. Find the eigenvalues of A. Processing continues only if A has one real ( $\lambda_R$ ) and a pair of complex-conjugate eigenvalues ( $\lambda_C$ ).
4. At each node of the tetrahedron, subtract the velocity component in the direction of the eigenvector corresponding to  $\lambda_R$ . This is equivalent to projecting the velocity onto the plane normal to the eigenvector belonging to  $\lambda_R$ , and can be expressed as

$$\vec{w} = \vec{u} - (\vec{u} \cdot \vec{n})\vec{n} \quad (3.2)$$

where  $\vec{n}$  is the normalized eigenvector corresponding to  $\lambda_R$ , and  $w$  is the reduced velocity.

5. Linearly interpolate each component of the reduced velocity to obtain

$$w_i = a_i + b_i x + c_i y + d_i z \quad (3.3)$$

$$i = 1, 2, 3$$

6. To find the center, set  $w_i$  in Eqn. (3.3) to zero. Since the reduced velocity lies in a plane, it has only 2 degrees of freedom. Thus, only 2 of the 3 equations in Eqn. (3.3) are independent. Any 2 can be chosen as long as their coefficients are not all zero.

$$\mathbf{0} = a_i + b_i x + c_i y + d_i z \quad (3.4)$$

$$i = 1, 2$$

which are the equations of 2 planes, whose solution (the intersection of 2 planes) is a line.

7. If this line intersects the cell at more than 1 point, then the cell contains a center of a local swirling flow. The center is defined by the line segment formed by the 2 intersection points.

Since the 2 intersection points lie on the line found in step 6, the reduced velocity at those points must be zero. This suggests a different (but equivalent) and more efficient way to finding the center. This approach renders steps 5, 6, and 7 unnecessary and replaces them with a new step 5:

5. For each of the tetrahedron's face, determine if there is exactly 1 point on the face where the reduced velocity is zero. If at the end there are exactly 2 distinct points, then the cell contains a center, which is defined by those 2 points.

Both approaches have been tried with identical results. Therefore, the second approach is implemented.

### 3.3 Testing

The algorithm is first tested on artificially-generated vector fields where the location of the center of the swirling flow is known exactly. The field is defined by

$$u = y - y_c, v = x - x_c, w = f(z) \tag{5}$$

Note that this vector field has circular streamlines (in the x-y plane) around a central axis whose location is defined by  $x_c$  and  $y_c$ . The magnitude of the vector is equal to the distance from the central axis. The field is discretized using an 11 x 11 x 11 node structured grid. The results for various values of  $x_c$ ,  $y_c$ , and functional forms of  $v_z$  (including constant, linear, and exponential) are studied and determined to be correct. A sample result (with  $x_c = 2.2 \Delta x$ ,  $y_c = 1.5 \Delta y$ , and  $v_z = 1$ ) is shown in Fig. 3.3. A streamline is also shown in this figure to provide a sense of the swirling vector field.

Table 3.1 Size of Test Cases for Swirl-Flow Finder

Algorithm

Case	Number of Nodes	Number of Cells	Number of Tetrahedral Cells*	Calculation Time (sec.)**
Cylinder	131072	123039	738234	34
F-117	48518	240122	240122	16

\* After decomposition (if needed) of original cells.

\*\* On SGI Indigo2 with MIPS R4400 150 Mhz CPU.

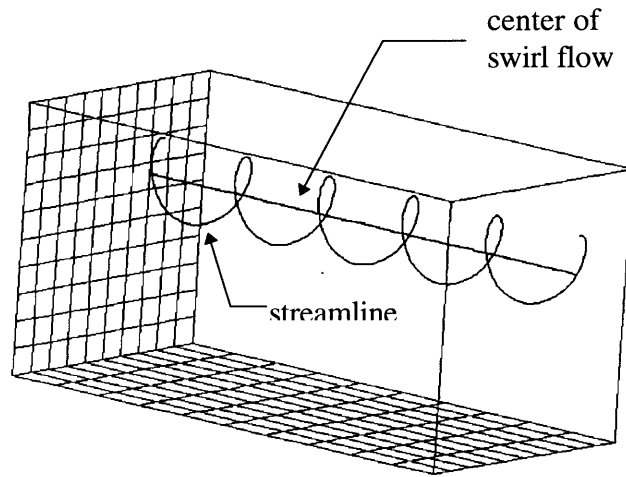


Figure 3.3 A sample result of an artificially-generated test case.

Further tests are done using data from 3-D calculations of flow past a tapered cylinder [26] and of flow over an F-117 fighter at an angle of attack [37]. The tapered-cylinder calculation employs structured grid, while the F-117 case uses unstructured grid composed of tetrahedra. The size of these data sets and the time needed to find the swirl flow centers are summarized in Table 3.1.

The results are shown in Figs. 3.4 and 3.5. To indicate the existence swirling flow, streamlines have been spawned near the centers found by the algorithm. These results indicate that the large coherent structures found by the algorithm do indeed correspond to centers of swirling flow. However, the algorithm does not find all the swirling flow in the tapered cylinder data. Missing are a few swirling flow structures further downstream of the cylinder, which are found by studying the vector field more closely. It is believed that the size of the grid cells might be a factor. The cells are larger away from the cylinder, reducing the accuracy in the calculation of the velocity-gradient tensor (and consequently the reduced velocity). Another possible cause is the algorithm's sensitivity to the strength of the swirl flow. As shown in Figs. 3.4b and 3.4c, the structures are very coherent for strong swirls (i.e., the swirl velocity is larger than or comparable to the normal velocity). However, the structures start to break up as the swirl weakens, and

further downstream, where the swirl flows are very weak, the algorithm finds no coherent structures.

In the case of the F-117 data, the structures are less coherent than in the tapered cylinder. The tetrahedral grid used in this data is very irregularly sized, and is rather coarse. Comparison between the streamlines in Figs. 3.4c and 3.5 also shows that the swirl in the F117 data is noticeably weaker. Both of these factors might contribute to the incoherence in the structures.

Comparisons have also been done between the results of this algorithm with that of FAST's vortex-core finder [14] [2]. FAST's finder defines a vortex core by integrating from a critical point in the direction of the eigenvector corresponding to the only real eigenvalue of the velocity-gradient tensor. For this comparison, 3 artificially-generated data sets are used, each bounded by a cube containing 3 randomly-placed vortices. The data generator is developed by D. Asimov at NASA Ames Research Center.

The comparisons are shown in Figs. 3.6a to 3.6f. Despite the lack of any 3-dimensional cues, the curves in these figures do exist in 3-D space, and each pair of figures are taken from the same view point. A high degree of similarities are found in each case except for the middle curves in each data set, where FAST produces curves that are either longer and/or has different orientation. Closer inspection of the data shows that pV3's results are the correct ones, while FAST's curve integrations veer away from the core due to the difficulty in integrating near critical points.

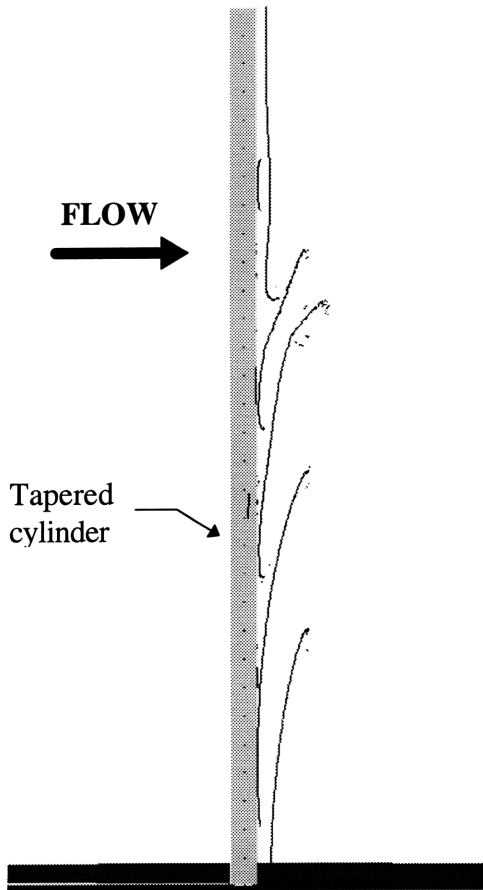


Figure 3.4a Flow past a tapered cylinder. Shown are swirl flow centers found by the algorithm.

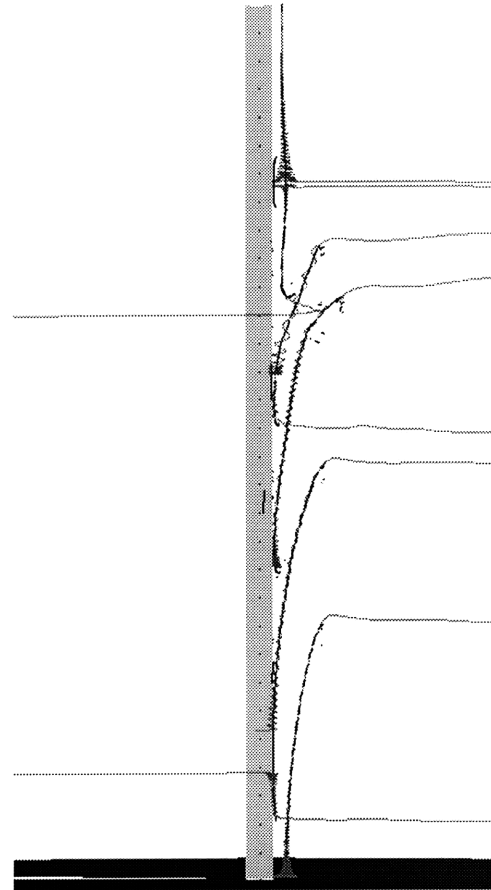


Figure 3.4b flow past a tapered cylinder. Swirl flow centers and streamlines are shown.

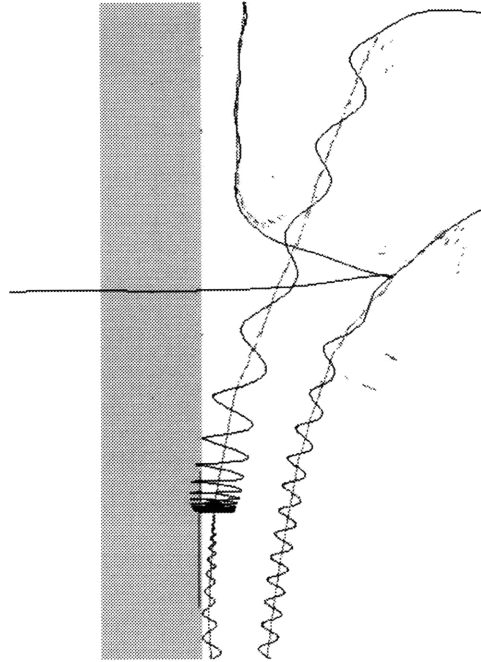


Figure 3.4c Blow up of figure 4b.

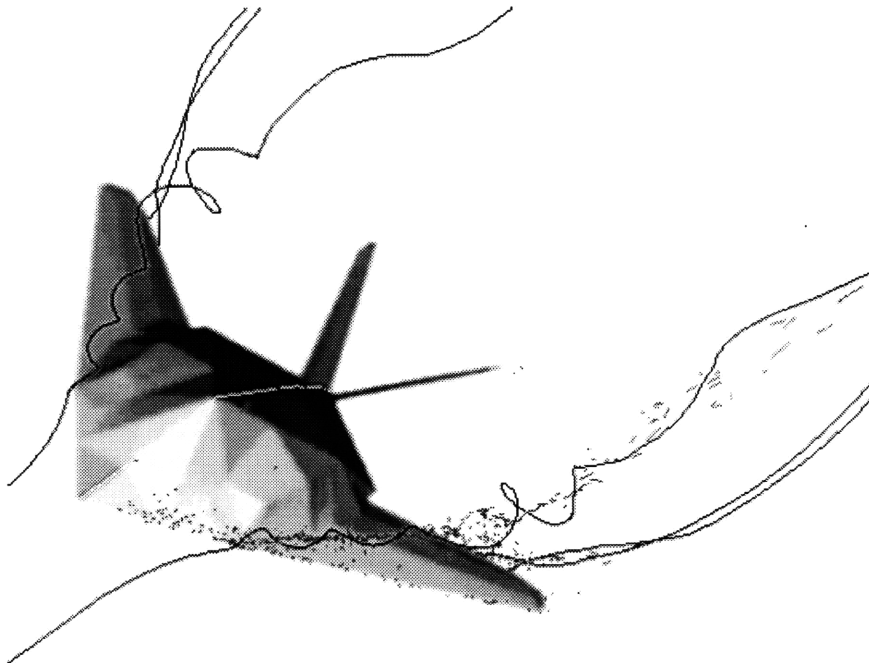


Figure 3.5 Flow over a F117 fighter. Swirl flow centers and streamlines are shown. Note: The centers are not mirrored.

Integration  
veers away

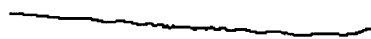



Figure 3.6a Result of FAST vortex core  
finder on data set 1.

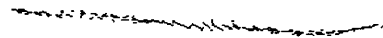


Figure 3.6b Result of pV3 swirl flow finder  
on data set 1.

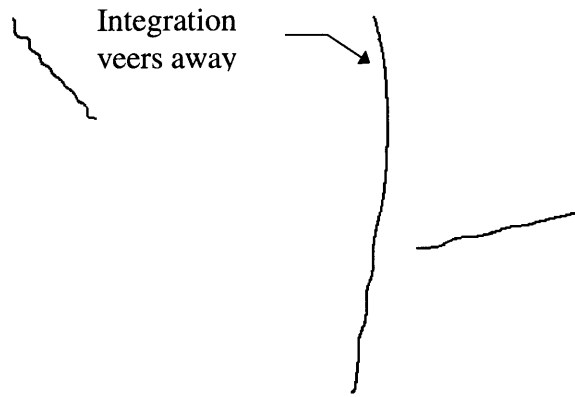
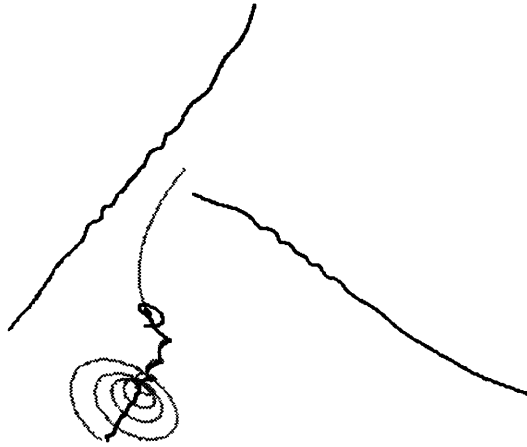


Figure 3.6c Result of FAST vortex core finder on data set 2.



Figure 3.6d Result of pV3 swirl flow finder on data set 2.





**Figure 3.6e** Result of FAST vortex core finder on data set 3. Note that a streamline has also been spawned here to indicate that the twist at the top of the middle curve is actually outside the vortex.



**Figure 3.6f** Result of pV3 swirl flow finder on data set 3.

## Chapter 4

# Residence Time

Flow separation represents interesting, and sometimes important, features in many types of flow calculations. In turbomachinery, separated flows are associated with extremely hot regions where high-speed hot flow exiting the combustor has been stagnated. These hot spots are very undesirable since the allowable operating stress of the turbine blades are closely related to temperature. In flow over a wing, the adverse pressure gradient on the wing upper surface can lead to separated flow. The major consequences of this phenomenon are a drastic loss of lift (or stalling) and a significant increase in pressure drag.

This interest in separated flow motivates the development of a tool which can automatically locate these regions. Ideally, the tool can work directly on the vector fields at each time slice of an unsteady data, without requiring other types of data or information from other time levels. An attempt was made to develop an algorithm, based on critical point theory, which works solely on the vector fields. However, this scheme was found to be unreliable in detecting separated flows. For documentation, the algorithm is described in Appendix A.

Helman and Hesselink [22] [23] have developed a visualization scheme for generating separation surfaces using only the 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 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. It is also unclear how a recirculation region should be defined in unsteady flows since a region that appears to be recirculating at a time slice might actually be moving with the flow as time progresses.

Discussions and communications with Haimes [15] and Giles [12] lead to the development of a scheme based on residence time. Essentially, this tool computes the amount of time the fluid has been in (or in residence within) the domain by integrating the residence-time governing equation over time. Time zero is defined as the time when the tool is turned on by the user. Thus, this tool provides information viewed in a frame of reference moving with the fluid, in contrast to streamlines and particle paths which present information viewed from a fixed reference frame.

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 it is considerably larger than that of the surrounding fluid. The iso-surface tool can then be used to distinguish this region. Therefore, residence time can potentially be used to locate separation regions.

The equations used for computing residence time will be discussed in the first section of this chapter. Section 2 describes how the equations are solved in discretized flow fields. Section 3 and 4 discuss the boundary conditions and the numerical smoothing, respectively. Some sample calculations are shown and discussed in section 5.

## 4.1 Residence time equation

The residence time of a fluid particle is defined by

$$\frac{D\tau}{Dt} = 1 \quad (4.1)$$

where  $\tau$  denotes residence time. And since  $\frac{D}{Dt} \equiv \frac{\partial}{\partial t} + \vec{u} \cdot \nabla$ , then Eqn. (4.1) becomes

$$\frac{\partial \tau}{\partial t} + \vec{u} \cdot \nabla \tau = 1 \quad (4.2)$$

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

$$\tau(x, y, z) = 0 \quad (4.3)$$

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

$$\tau(x, y, z) = 0 \quad \text{at inflow} \quad (4.4)$$

To obtain the conservative form of Eqn. (4.2) for incompressible flow, Eqn (4.2) can be rewritten as

$$\begin{aligned} \frac{\partial \tau}{\partial t} + \vec{u} \cdot \nabla \tau + \tau \nabla \cdot \vec{u} &= 1 + \tau \nabla \cdot \vec{u} \\ \frac{\partial \tau}{\partial t} + \nabla \cdot \tau \vec{u} &= 1 + \tau \nabla \cdot \vec{u} \end{aligned}$$

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

$$\frac{\partial \tau}{\partial t} + \nabla \cdot \tau \vec{u} = 1 \quad (4.5)$$

The conservative form for compressible flow can be obtained by rewriting Eqn. (4.2) as

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

where  $\rho$  denotes density. And since the conservation of mass equation for compressible flow is

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

then

$$\begin{aligned} \rho \frac{\partial \tau}{\partial t} + \vec{u} \cdot \rho \nabla \tau + \tau \left[ \frac{\partial \rho}{\partial t} + \nabla \cdot \rho \vec{u} \right] &= \rho \\ \frac{\partial \rho \tau}{\partial t} + \nabla \cdot (\rho \tau \vec{u}) &= \rho \end{aligned} \quad (4.6)$$

The effect of viscosity on  $\tau$  is similar to its affect on velocity because the same mechanism is at work in both instances. Therefore, the viscous term for Eqn. (4.6) is analogous to 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

$$\frac{\partial \rho \tau}{\partial t} + \nabla \cdot (\rho \tau \bar{u}) = \rho + \nabla \cdot \mu \nabla \tau$$

or

$$\frac{\partial \rho \tau}{\partial t} + \nabla \cdot (\rho \tau \bar{u} - \mu \nabla \tau) = \rho \quad (4.7)$$

where  $\mu$  is the absolute viscosity, which accounts for both laminar and turbulent viscosity.

For a flow with constant viscosity and density, Eqn. (4.7) reduces to

$$\begin{aligned} \frac{\partial \tau}{\partial t} + \nabla \cdot \tau \bar{u} &= 1 + \frac{\mu}{\rho} \nabla^2 \tau \\ \frac{\partial \tau}{\partial t} + \nabla \cdot \left( \tau \bar{u} - \frac{\mu}{\rho} \nabla \tau \right) &= 1 \end{aligned} \quad (4.8)$$

where  $\frac{\mu}{\rho}$  is constant.

All the conservative forms of the residence-time equation (i.e. Eqns. (4.5), (4.6), (4.7), and (4.8)) can be expressed as

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} + \frac{\partial H}{\partial z} = Q \quad (4.9)$$

where, for incompressible inviscid flow

$$U = \tau, \quad F = \tau u, \quad G = \tau v, \quad H = \tau w, \quad Q = 1,$$

for compressible inviscid flow

$$U = \rho \tau, \quad F = \rho \tau u, \quad G = \rho \tau v, \quad H = \rho \tau w, \quad Q = \rho,$$

for compressible viscous flow

$$U = \rho \tau, \quad F = \rho \tau u - \mu \frac{\partial \tau}{\partial x}, \quad G = \rho \tau v - \mu \frac{\partial \tau}{\partial y}, \quad H = \rho \tau w - \mu \frac{\partial \tau}{\partial z}, \quad Q = \rho$$

and for a flow with constant viscosity and density

$$U = \tau, \quad F = \tau u - \frac{\mu}{\rho} \frac{\partial \tau}{\partial x}, \quad G = \tau v - \frac{\mu}{\rho} \frac{\partial \tau}{\partial y}, \quad H = \tau w - \frac{\mu}{\rho} \frac{\partial \tau}{\partial z}, \quad Q = 1$$

Note that Eqn. (4.9) has a form similar to the conservative formulation of the Euler equations. This similarity enables the use of an Euler solver developed in another work. The algorithm, which will be discussed in the next section, operates on a cell-by-cell manner, and, therefore, can readily take advantage of pV3's parallel capability.

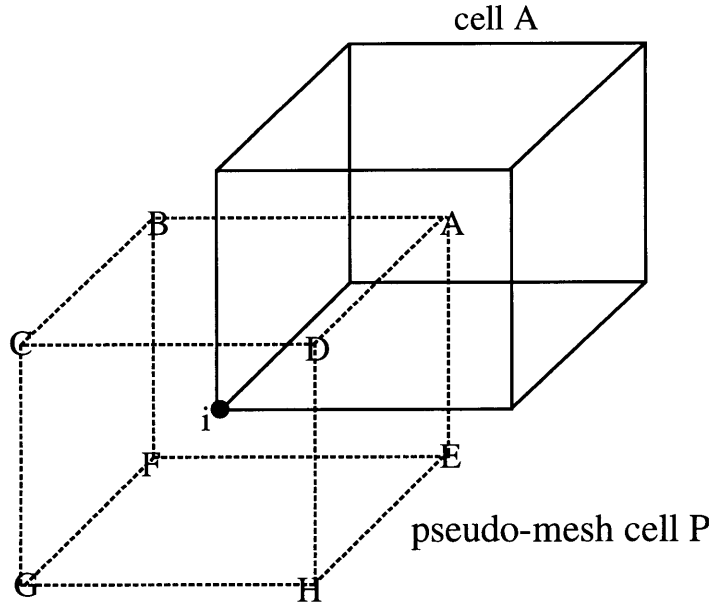


Figure 4.1 Pseudo-mesh cell P (dashed) surrounding node  $i$  and a cell neighbor A. The corners of P are composed of the center of neighboring cells marked by capital letters.

## 4.2 Lax-Wendroff algorithm

An explicit time-marching algorithm of Lax-Wendroff type is used to solve Eqn. (4.9). This scheme is identical to that used by Saxer [34] to solve the Euler equations for a stator/rotor flow. The basic integration scheme is similar to the one introduced by Ni [29], recasted by Hall [21], and then extended by Ni and Bogoian [30] to 3-D. Saxer [34] then adapted the formulation to handle unstructured grids, particularly unstructured hexahedral cells.

In solving the residence time equation, it is assumed that the flow variables  $u$ ,  $v$ ,  $w$ ,  $\rho$ , and  $\mu$  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 corner nodes. The flux residual is computed by adding the fluxes through the six faces, and then adding the source term for the cell. This residual is then distributed to the eight corner nodes according to the Lax-Wendroff algorithm to evaluate the residence time change at those nodes. The details of the construction of the algorithm are

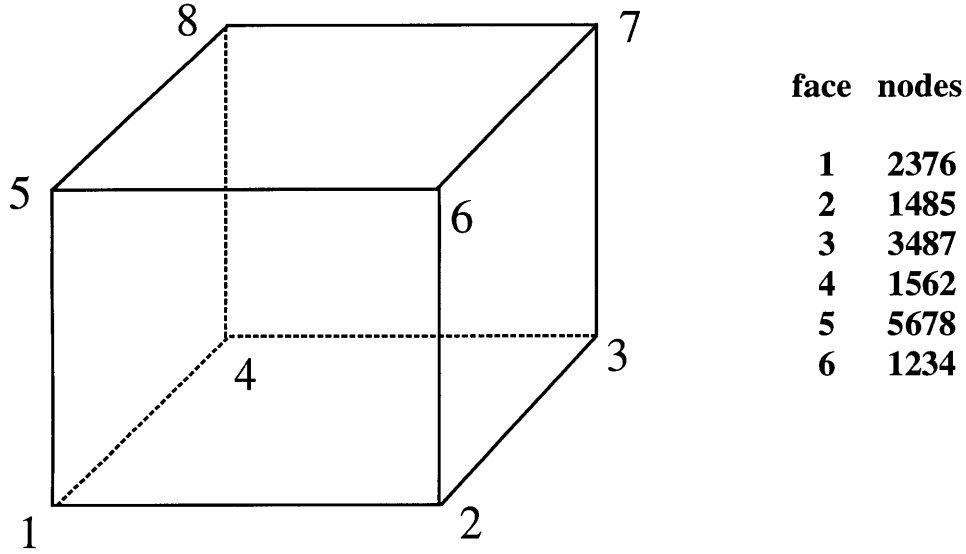


Figure 4.2 Face and node numbering for hexahedral cell.

discussed in [34]. The construction is repeated in Appendix B for convenience. However, the final results can be presented here without loss of continuity.

A pseudo-mesh cell P can be constructed around node  $i$ . The corners of cell P consist of the centers of cells which has node  $i$  as one of its node, as illustrated in Fig. 4.1. The node and face numbering of a hexahedral cell is as shown in Fig. 4.2. Then, the contribution of one of the adjacent cell, say cell A, to the change at node  $i$  is

$$\delta U_{iA} = \frac{1}{8} \left( \frac{\Delta t}{V} \right)_i \left\{ \left( \frac{V}{\Delta t} \right)_A \Delta U_A - \sum_{f=1,3,5} (\Delta F_A \bar{S}_x + \Delta G_A \bar{S}_y + \Delta H_A \bar{S}_z)_f + \frac{V_A}{2} \Delta Q_A \right\} \quad (4.10)$$

where, for inviscid incompressible flow,

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

for inviscid compressible flow,

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

for viscous compressible flow,

$$\begin{aligned}\Delta F &= \bar{u} \Delta(\rho\tau) + (\overline{\rho\tau}) \Delta u - \bar{\mu} \Delta \left( \frac{\partial \tau}{\partial y} \right) - \left( \overline{\frac{\partial \tau}{\partial x}} \right) \Delta \mu, \\ \Delta G &= \bar{v} \Delta(\rho\tau) + (\overline{\rho\tau}) \Delta v - \bar{\mu} \Delta \left( \frac{\partial \tau}{\partial y} \right) - \left( \overline{\frac{\partial \tau}{\partial y}} \right) \Delta \mu, \\ \Delta H &= \bar{w} \Delta(\rho\tau) + (\overline{\rho\tau}) \Delta w - \bar{\mu} \Delta \left( \frac{\partial \tau}{\partial y} \right) - \left( \overline{\frac{\partial \tau}{\partial y}} \right) \Delta \mu, \\ \Delta Q &= \Delta \rho\end{aligned}$$

for flow with constant density and viscosity,

$$\begin{aligned}\Delta F &= \bar{u} \Delta(\rho\tau) + (\overline{\rho\tau}) \Delta u - \left( \frac{\bar{\mu}}{\rho} \right) \Delta \left( \frac{\partial \tau}{\partial y} \right), \\ \Delta G &= \bar{v} \Delta(\rho\tau) + (\overline{\rho\tau}) \Delta v - \left( \frac{\bar{\mu}}{\rho} \right) \Delta \left( \frac{\partial \tau}{\partial y} \right), \\ \Delta H &= \bar{w} \Delta(\rho\tau) + (\overline{\rho\tau}) \Delta w - \left( \frac{\bar{\mu}}{\rho} \right) \Delta \left( \frac{\partial \tau}{\partial y} \right), \\ \Delta Q &= 0\end{aligned}$$

and

$$\Delta U_A = - \left( \frac{\Delta t}{V} \right)_A \left( \sum_{f=1}^{6 \text{ faces}} \bar{F} S_x + \bar{G} S_y + \bar{H} S_z \right)_{cell A} + \Delta t_A Q_A$$

Except for quantities defined above, the subscript A denotes quantities evaluated at cell A, while the subscript  $i$  stands for average quantities at P, the pseudo-cell associated with node  $i$ . The bar over  $F$ ,  $G$ , and  $H$  denotes an average over the four grid nodes of the face  $f$ .  $S_x$ ,  $S_y$ , and  $S_z$  are the projected areas on the  $yz$ ,  $xz$ , and  $xy$  planes of the face  $f$ .  $\bar{S}_x$ ,  $\bar{S}_y$ , and  $\bar{S}_z$  refer to the averaged projection areas of opposite faces of cell A. Other quantities with overbar are averages over the cell.



The contributions to node  $i$  from cells B, C, D, E, F, G, and H are computed in a similar manner. The sum of these contributions define the change at node  $i$ . Thus,

$$\delta U_i = \sum_{j=1}^{8 \text{ cells}} \delta U_{ij} = \delta U_{iA} + \delta U_{iB} + \delta U_{iC} + \delta U_{iD} + \delta U_{iE} + \delta U_{iF} + \delta U_{iG} + \delta U_{iH} \quad (4.11)$$

As discussed by Saxer [34], this scheme is conservative because the discrete solution approaches the analytic solution as the grid is refined. It is spatially second-order accurate even for non-uniform grids, and also second-order accurate in time.

### 4.3 Boundary conditions

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

$$U_i = \mathbf{0}, \quad \text{for all node } i \text{ at the inflow boundary.}$$

At outlet boundaries, the simplest alternative has been used, which is to do nothing. Essentially, this approach assumes that the contributions to  $\delta U_i$  by the cells just upstream of the boundary and by the non-existent cells just downstream of it are equal. This assumption is reasonable as long as the gradients of the flow quantities in the direction normal to the boundary is small. However, if the user of this tool determines that such an assumption no longer applies or a more elaborate boundary treatment would be more appropriate, a subroutine can be supplied which overrides the standard method. The values of  $U$  and  $\delta U_i$  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 algorithm must ensure that there is no flux through the wall. This is accomplished by making a correction to the nodes of cells adjacent to the wall boundary. The correction is performed after the Lax-Wendroff changes  $\delta U$  have been distributed to all nodes, including the nodes at the wall boundaries. The contribution of cell A (A is adjacent to the wall) to  $\delta U_{iA}$  is corrected as follows

$$(\delta U_{iA})_{wall} = (\delta U_{iA})_{field} + \frac{1}{8} \left( \frac{\Delta t}{V} \right)_i (\Delta U_A)_{wall\ correction} \quad (4.12)$$

where

$$(\Delta U_A)_{wall\ correction} = (\bar{F}S_x + \bar{G}S_y + \bar{H}S_z)_{f=wall\ face} \quad (4.13)$$

This correction is performed on the eight nodes of cell A. Note that the wall correction, Eqn. (4.13), is simply the flux error introduced in the field calculation.

#### 4.4 Numerical smoothing

A fourth-difference smoothing operator identical to the one used by Saxer [34] is added to the Lax-Wendroff scheme to damp out non-physical oscillations that can be introduced by the scheme. The smoothing term is added to the right-hand-side of Eqn. (4.10) and has the form

$$-(SF)_{smo4} \nabla \cdot (l \nabla (l^2 \nabla^2 U))$$

where  $(SF)_{smo4}$  is a scaling factor, and  $l$  is a length comparable to the local grid size. In discrete form, the smoothing operator becomes

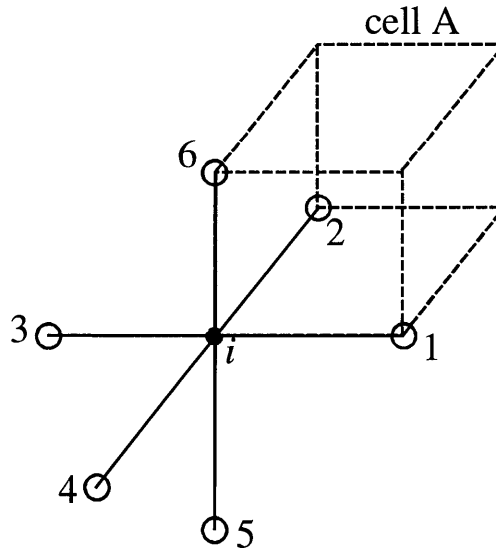


Figure 4.3 Stencil for pseudo-Laplacian.  
Only cell A is shown.

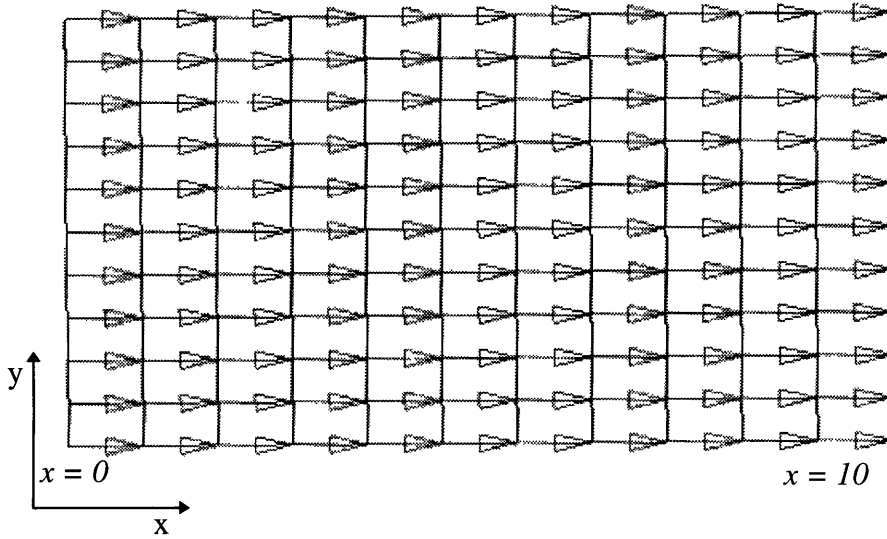


Figure 4.4a Artificially-generated flow with constant velocity in the x direction,  $v_x = 1$ .

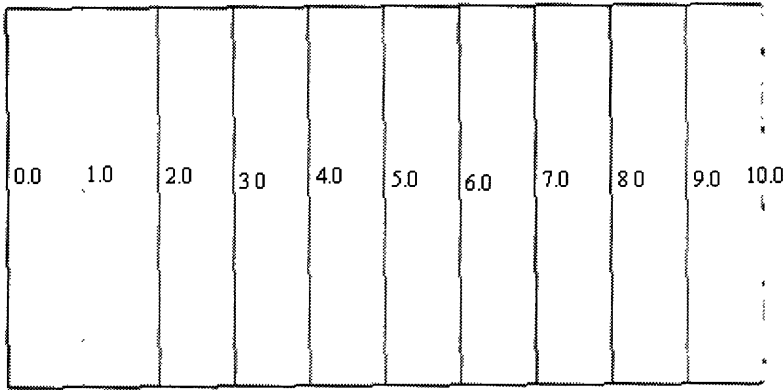


Figure 4.4b Residence-time contours for flow shown in Fig. 4.4a

$$(\delta U_i)_{smo4} = \left( \frac{1}{V} \right) \sum_{j=1}^{8 \text{ cells}} -v V_j (\bar{D}_j^2 - D_i^2)$$

$D_i^2$  is a pseudo-Laplacian based on the six edge nodes surrounding node  $i$ , as shown in Fig. 4.3.

It is defined by Holmes and Connell [24] as

$$D_i^2 = \sum_{k=1}^{6 \text{ nodes}} \psi_k (U_k - U_i)$$

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

$$\bar{D}_j^2 = \sum_{k=1}^{8 \text{ corner nodes}} \frac{1}{8} D_k^2$$

$\Psi_k$  is a grid-dependent weight which determines the degree of dependence on the neighboring nodes. For details on how this value is obtained, the reader is referred to [34].  $\nu$  is a coefficient with a typical value of 0.002 to 0.01. A value of 0.005 is used for the calculations shown in the next section.

## 4.5 Sample results

This algorithm is first tested using artificially-generated data sets to ensure that the computed residence time agrees with the easily-computed analytical solution. A sample data set is shown in Fig. 4.4a, in which the flow has a constant velocity in the x direction. In this case, the algorithm for inviscid incompressible flow is used. A contour plot of the result is shown in Fig. 4.4b, which agrees well with the expected result for such uniform flow.

The algorithm has also been tested on a flow through a converging-diverging duct, shown in Fig. 4.5a, computed by Darmofal [11] using the incompressible, axisymmetric Navier-Stokes equations. In this flow, a separation bubble (which correspond to a streamfunction value of 0) exists just downstream of the converging section, shown in Fig. 4.5a as an iso-surface of streamfunction with value 0. The viscous incompressible residence-time algorithm is employed here, and the results is shown as a contour plot in Fig. 4.5b. The residence time has been non-dimensionalized by dividing by the vortex core size and multiplying by the freestream velocity [11]. Note the elliptical region of high residence time fluid downstream of the constriction as well

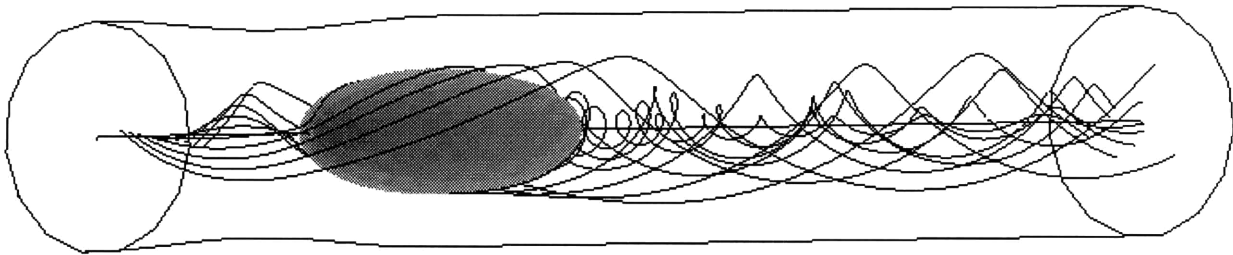


Figure 4.5a Swirling flow through a converging-diverging duct [11]. A separation bubble is shown as an iso-surface of streamfunction with value 0. Streamlines are also shown.

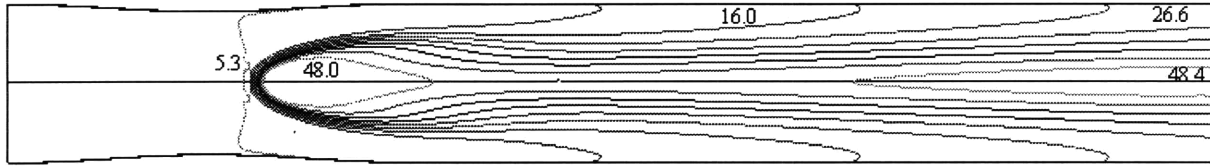


Figure 4.5b Contours of residence time for flow through a converging-diverging duct.

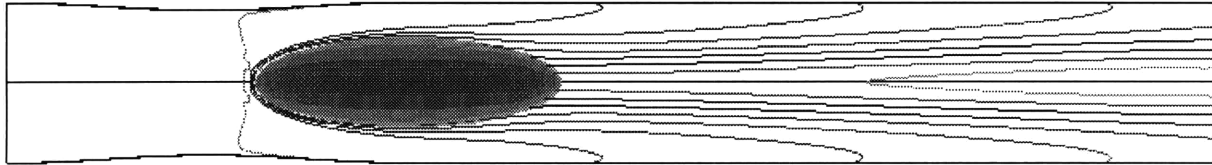


Figure 4.5c Contours of residence time and iso-surface of streamfunction with value 0.

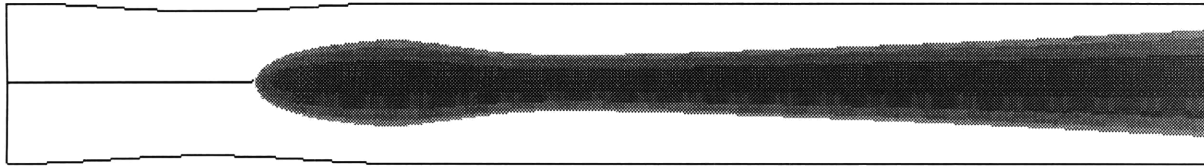


Figure 4.5d Iso-surface of residence time of value 27.5.

as at the wake of the ellipse. The elliptical region corresponds to the fluid “trapped” within the separation bubble. The wake contains high residence time fluid because the fluid that ends up in the wake travels through a slow-moving path around the separation bubble, as can be seen in Fig. 4.5a. Fig. 4.5c combines the residence-time contour and the streamfunction iso-surface. Fig. 4.5d show the iso-surface of residence time of value 27.5. The elliptical upstream portion of this iso-surface closely matches the separation bubble. The rear section of the ellipse is somewhat thinner compared to the separation bubble due to the effect of viscosity, which mixes the high and low residence-time fluid.

To demonstrate its applicability for multi-partitioned data, the algorithm is applied on an unsteady stator/rotor flow computed by Saxer [34]. In the examples shown here, the computation solves the 3-D unsteady Euler equations on a  $40 \times 15 \times 15$  grid for both the stator

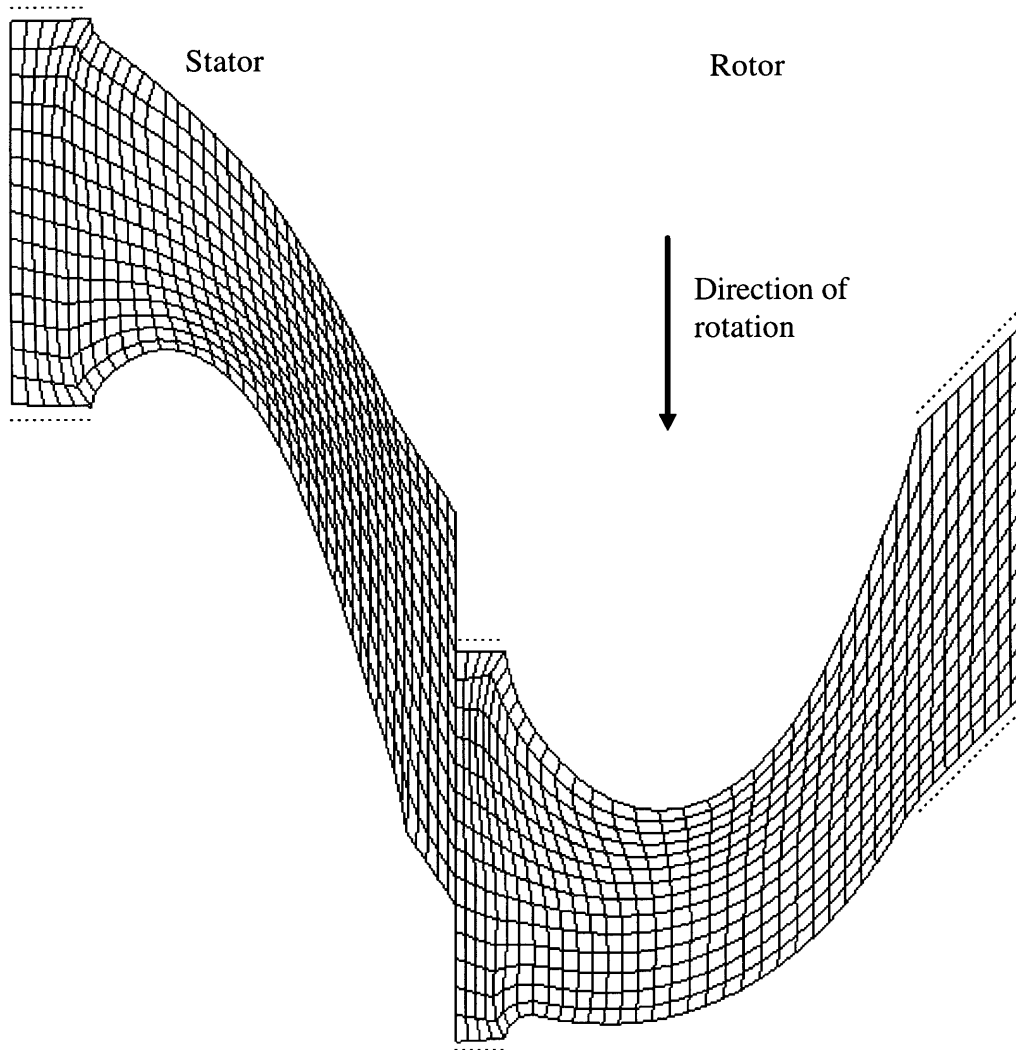


Figure 4.6a Mesh at the hub of the stator/rotor (stator: 40 x 15 x 15, rotor: 40 x 15 x 15). Dashed line segments indicate periodic boundaries.

and rotor passages. The data and computation for the stator and rotor are handled by two separate processes. The size of the computational domain, illustrated in Figs. 4.6, has been reduced from multiple blade passages to one blade-to-blade passage with a stator-to-rotor pitch ratio of 1. The data is visualized in a post-processing manner by interpolating from 10 equal-interval time slices obtained in one blade-to-blade period.

Since the flow is assumed to be circumferentially periodic, some of the computational boundaries shown in Figs. 4.6 represent periodic boundaries. This information can be communicated to the residence-time module by defining an array containing the nodes that are

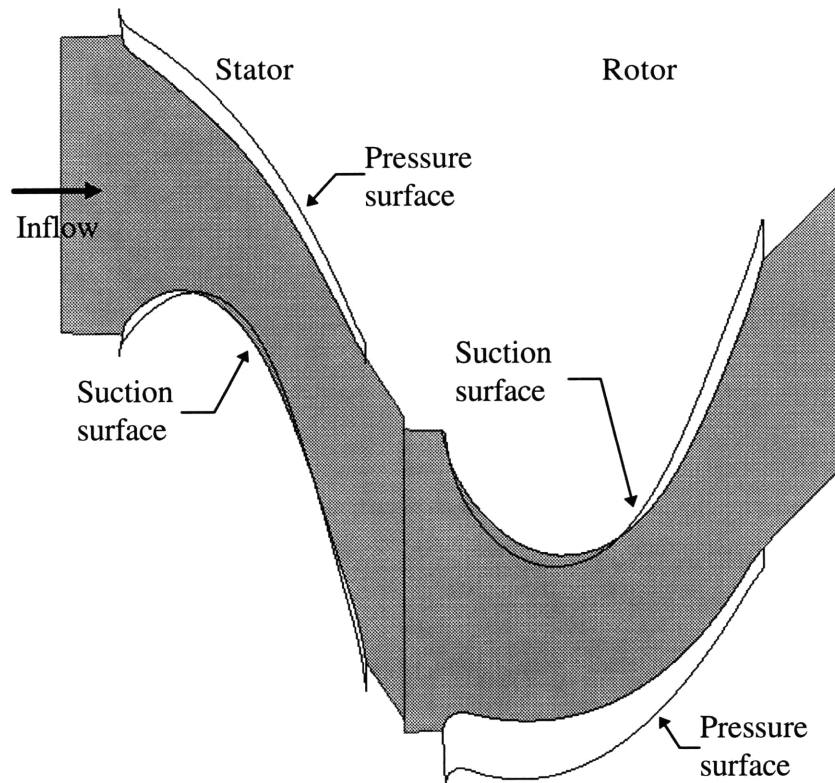


Figure 4.6b 3-D illustration of stator/rotor passage. Grey area is the hub.

periodic (or equivalent). Thus, if nodes 2 are defined to be equivalent, the changes for both nodes are summed and the sum, which represent the actual  $\delta U_i$ , is used to update the values at both nodes.

The stator/rotor interface in this computation is a special type of boundary due to the relative movement between the stator and the rotor. Thus, nodes at the stator outlet and at the rotor inlet are not necessarily aligned in the tangential direction. However, the gridding always ensures alignment in the radial direction. In order to deal with special cases such as this, pV3 allows the user to supply a subroutine in which the values of  $U$  and  $\delta U_i$  can be modified before the node values are updated. Within this subroutine the quantities ( $U$  and  $\delta U_i$  at the interface) needed to compute the changes at the stator/rotor interface are transmitted from one client to the other.

The way the changes are computed for this particular example can be best explained through an example. Consider nodes a, b, and c at the stator outlet and nodes o, p, and q at the rotor

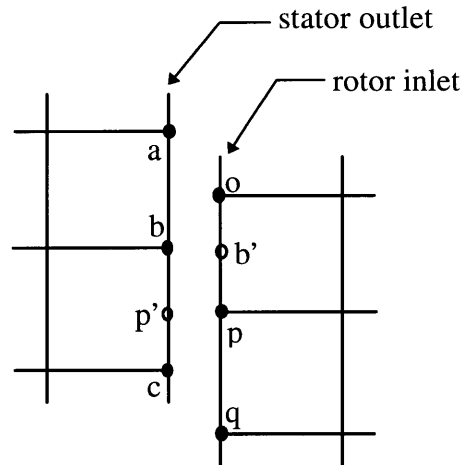


Figure 4.7 Nodes at the stator/rotor interface. Solid circles indicate actual nodes. Unfilled circles indicate locations corresponding to actual nodes at the opposite interface.

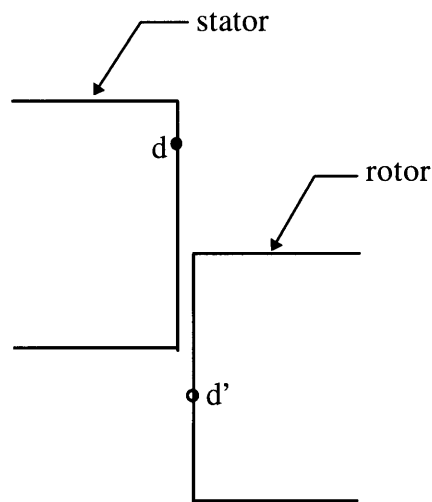


Figure 4.8 Nodes at non-overlapping region can be mapped to location at the opposite interface by using the periodicity assumption.

inlet, as shown in Fig. 4.7. All of these nodes are in the same radial plane. The change at node b is computed as the average of the change at b and the change at the corresponding point at the rotor inlet, b'. The change at b' is computed by linearly interpolating the changes at o and p. Thus,

$$\delta U_b = \frac{1}{2}(\delta U_{b,old} + \delta U_{b',old})$$



where the subscript “old” refers to quantities before modifications. The changes at other points at the stator outlet are computed in a similar manner. By assuming circumferential periodicity, it is possible to determine the change at the corresponding rotor inlet even though the surfaces do not entirely overlap. For instance, in Fig. 4.8 node d’ corresponds to node d.

The change at node p, at the rotor inlet, is defined such that the new values of  $U$  (i.e., after the updating) at p and at its corresponding point p’ will be equal. Thus,

$$\delta U_p = U_{p'} - U_{p,old}$$

where the subscript “old” refers to quantities before updating.  $U_{p'}$  is computed by linearly interpolating the new values of  $U$  at b and c. As in the stator outlet, the assumption of periodicity is also used to deal with non-overlapping regions.

The results are shown as residence-time contours in Figs. 4.9. The residence time has been non-dimensionalized by dividing by the stator blade axial chord at the hub and multiplying by inlet stagnation speed of sound. Note the high-residence time fluid at the wakes of the stator and rotor blades. This is due to the slower-moving fluid at the pressure surfaces. The movement of the rotor and the difference in the residence time of the fluid at the pressure and suction surface also result in the alternating pattern of low and high values in the rotor passage.

Fig. 4.10 displays the iso-surface of residence time of value 7.7. Here it can be seen clearly the wake region of the stator blades as well as some of the alternating structure in the rotor. The residence-time iso-surface of value 11.0 is shown in Fig. 4.11. The wake structure of the rotor blade is clearly shown. An interesting aspect is the existence of the surface at the pressure side near the stator trailing edge. What happens to this high residence time fluid when it flows downstream? Shouldn’t there be fluid with residence time much higher than 11 in the rotor passage? The fact that there is no region with values higher than 13 (anywhere in the domain) seems to suggest high residence time fluid is being “lost”. However, the answer lies in the realization that the value at a node represent an average for the fluid around the node. As the high-value fluid at the pressure surface flows downstream past the trailing edge it becomes mixed with the low-value fluid from the suction surface. This averaging lowers the high values. Closer inspection shows a similar occurrence at the rotor trailing edge.

These results also shows 3 regions where the distribution of residence time seems unrealistic. These regions are shown in Fig. 4.9a as regions A, B, and C. Fig. 4.12 displays an enlarged view of region B. Although the anomalous behavior are not readily obvious in Figs. 4.9 (particularly at A and C), they are apparent during the startup period. At the start of the residence time integration, the value at all nodes are set to zero. As time progresses, new fluid (which by definition has a residence time of zero) enters through the stator inlet. Thus, there is a wave front separating the fluid which exists at the time the integration starts and the fluid which enters afterwards. Until this wave front exits through the rotor outlet, any fluid downstream of it must have uniform residence time. Any non-uniformity represent anomalous behavior.

Fluid in regions A, B, and C is found to show significant non-uniformity at startup, on the order of  $\pm 10\%$  of the expected value. There are also strong correlation between the location of these regions and supersonic regions with large gradients in mach number and/or density. Figs. 4.13 show the mach number and density contours and the locations with large gradient. At this

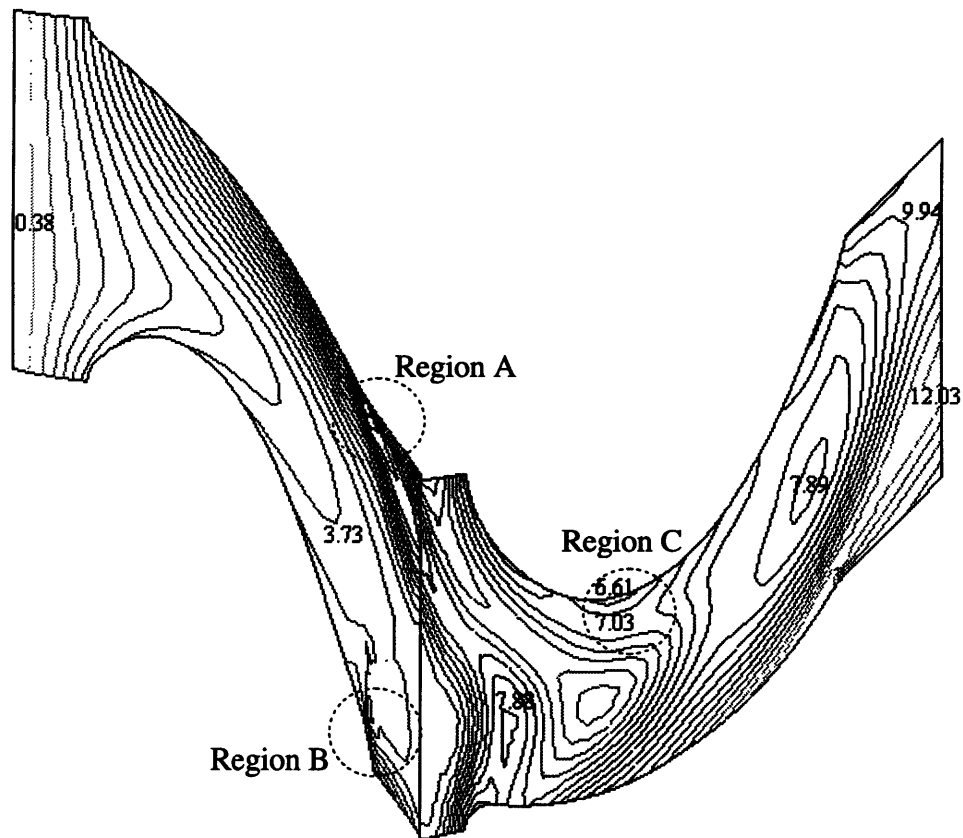


Figure 4.9a Residence-time contours at the hub. Regions where anomalies occur are indicated by dashed circles.

point, it is suspected that the anomalies are due to dispersive errors in the flow solution at these regions. Shapiro [35] performed a linearized analysis of the 2-D Euler equations and apply it to a number of numerical schemes, one of them being a cell-vertex algorithm similar to the one used by Saxer to compute the stator/rotor flow. Shapiro concluded that the low frequency oscillations occurring near regions of high gradients are due to dispersion in the numerical schemes. However, the connection between dispersive errors and the anomalies observed in the residence-time computation is still inconclusive and warrants further investigation.

Nevertheless, the existence of these anomalies does not undermine the purpose of this particular example. The applicability of the residence-time tool on multi-partitioned data has been demonstrated. And the ability to modify  $\delta U$  through a programmer-supplied subroutine is found to be very useful in dealing with special cases such as the one presented at the stator/rotor interface.

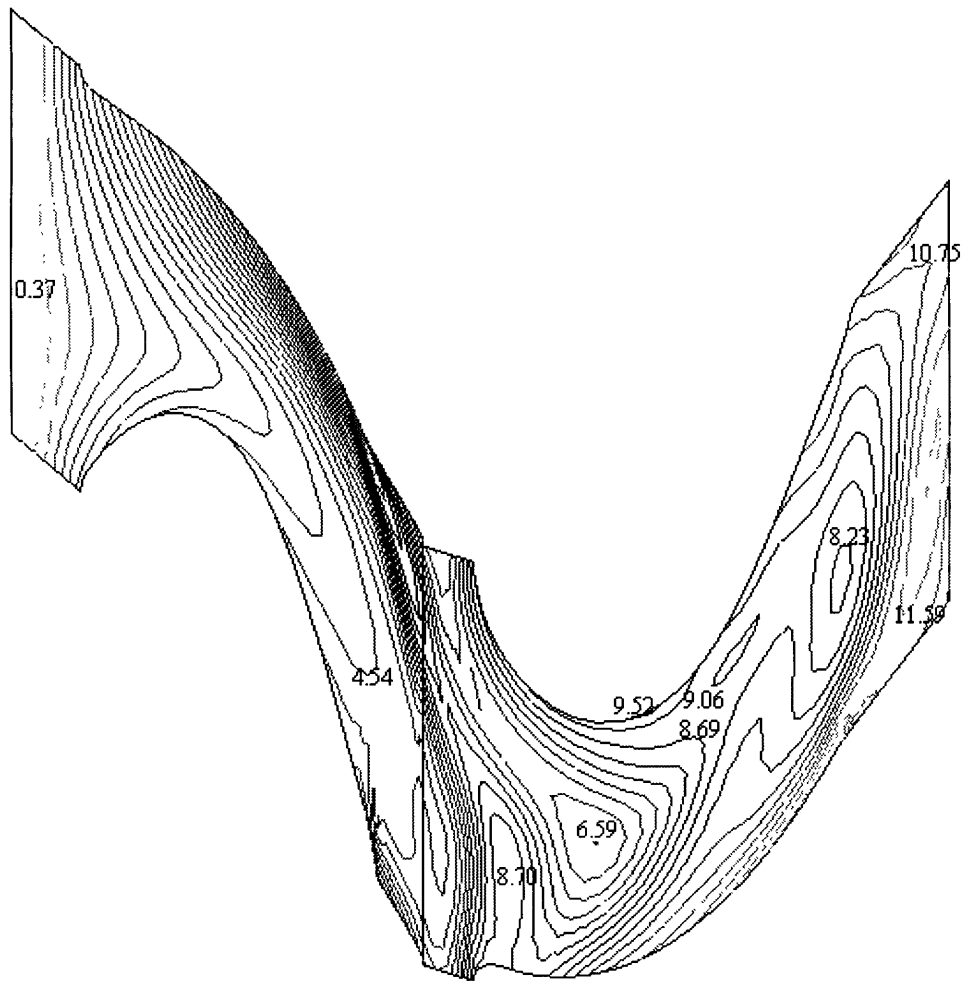


Figure 4.9b Residence time contours at the tip. Regions with anomalies are similar to the ones shown in Fig. 4.9a.

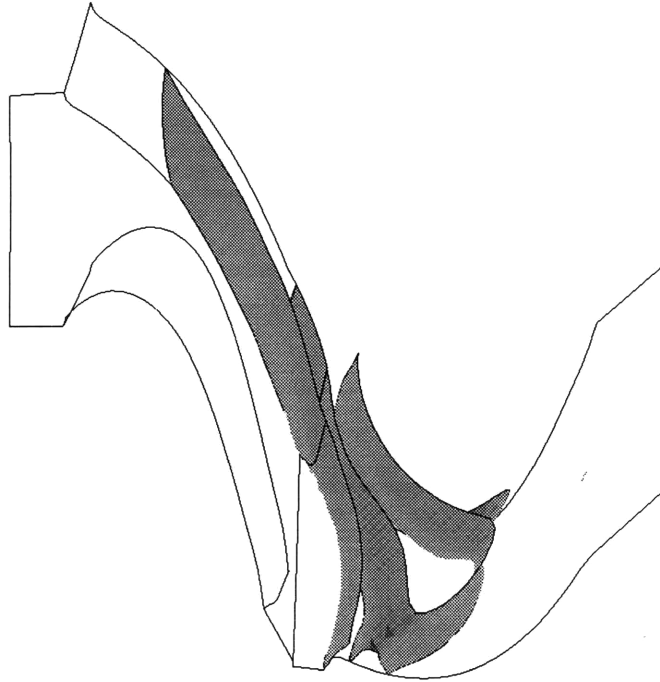


Figure 4.10 Residence time iso-surface of value 7.7. For clarity, rotor blade surfaces are not shown.

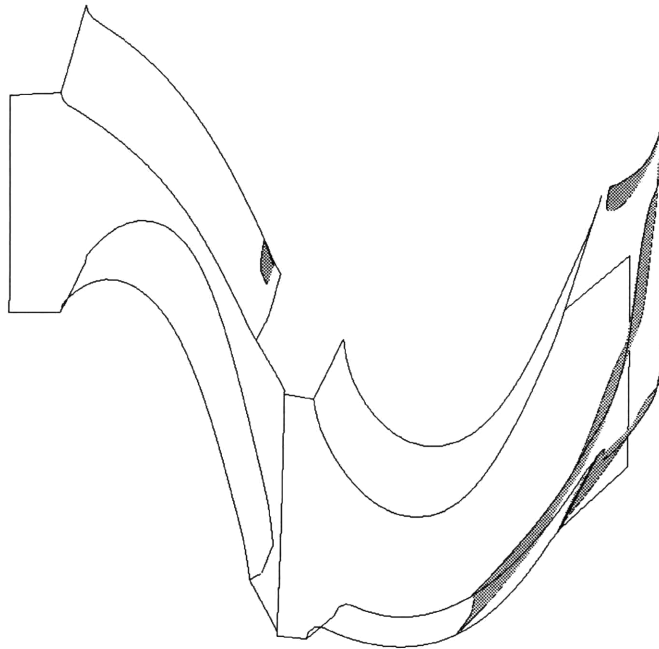


Figure 4.11 Residence time iso-surface of value 11.0.



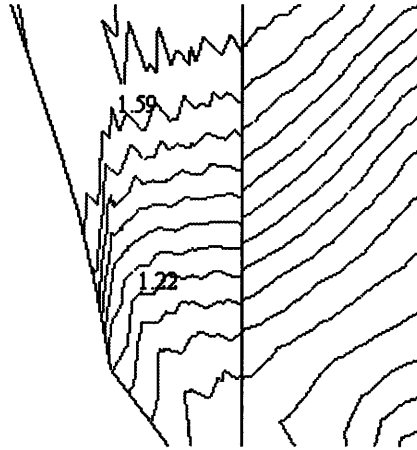


Figure 4.13b Mach number contours. Region (at the stator suction surface near the trailing edge) with large gradient.

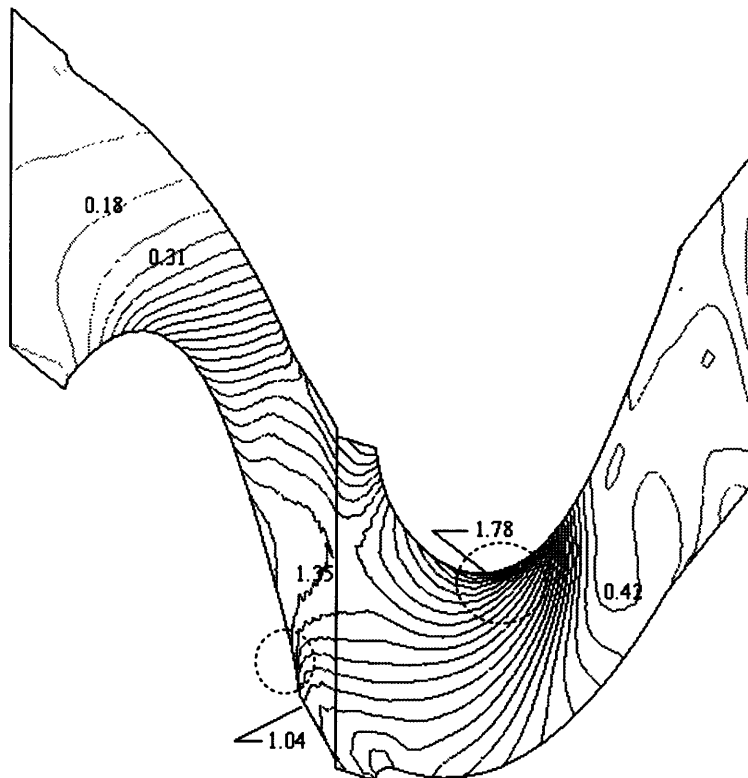


Figure 4.13c Mach number contours at the tip. Regions with large gradient indicate by dashed circles.

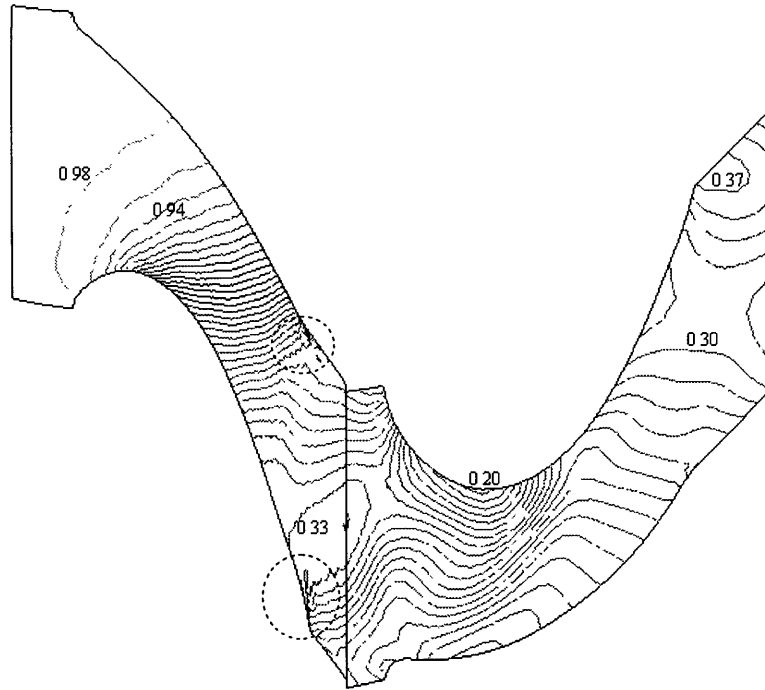


Figure 4.13d Density contours. Density non-dimensionalized by the inlet stagnation density. Regions with large gradient indicated by dashed circles.

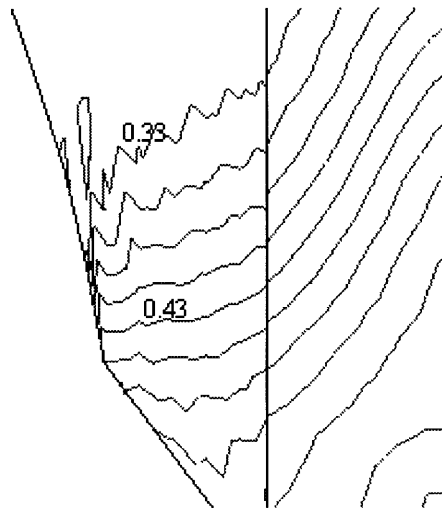


Figure 4.13e Density contours. A region with large density gradient (at the stator suction surface near the trailing edge).



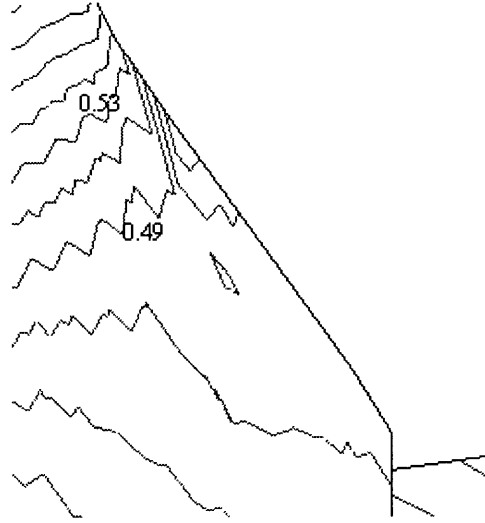


Figure 4.13f Density contours. A region with large density gradient (near the trailing edge of the stator pressure surface).

# Chapter 5

## Conclusion

### 5.1 Summary of Results

This thesis has addressed three primary topics which will be reviewed here. First, a method has been designed for managing the information movement needed to continue streamline and particle-path integrations across the internal boundaries of a multi-partition computational domain. Although the underlying structures of a streamline (a curve) and a particle (a point) are completely different, the similarity in continuing their integrations across internal boundaries has been exploited to arrive at a scheme that works in both cases. The scheme manages the flow of information between the clients, where the integrations are done, and the server, which does the rendering, in order to minimize network traffic, avoid multiple instances of the same object, and make efficient use of the parallel environment. It also provides a simple test to determine whether more integration transfers will be done within the current time-step. This knowledge enables the server to know when it can safely instruct the clients to proceed to the next time step.

Second, an algorithm to automatically locate the center of swirling flow in 3-D vector fields has been developed and implemented as part of the pV3 visualization package. By employing cell-by-cell processing and using only tetrahedral cells (and transforming other cell types to tetrahedra), the scheme can take advantage of pV3's distributed environment and the simplification from using linear interpolation. There is evidence suggesting that the strength of the swirl flow and the coarseness of the grid might affect the degree of accuracy and coherency of the results. However, tests using artificially-generated vector fields and 2 different CFD data have shown that the coherent structures found by the algorithm are indeed centers of swirling flow. Results on artificially-generated data have also been compared with those from the vector-field topology module of FAST [14] [2]. High degrees of similarities have been found. This

comparison also demonstrates the advantages of the algorithm over a curve-integration method used by FAST. There are difficulties in finding critical points and in identifying swirling flows whose critical points are outside the computational domain. Furthermore, a curve integration numerically tends to veer away from the core of the swirling flow, and is less able to take advantage of a parallel capability.

Finally, interest in flow separation has motivated the development of a residence-time integration tool. This tool computes the amount of time the fluid has been in (or in residence within) the computational domain. Since the fluid in separated regions remains there for a considerable amount of time, there will usually be a significant difference between the residence time of the fluid within the separated flow and that of the surrounding fluid. The iso-surface tool can then be used to identify this region.

The residence-time equation for different types of flows (inviscid incompressible, viscous incompressible, inviscid compressible, and constant density/viscosity) has been formulated. An explicit time-marching algorithm of Lax-Wendroff type is used to solve the residence time equation. The algorithm performs the computation on a cell-by-cell manner, and thus can readily take advantage of pV3's distributed processing. In order to handle the variety of possible boundary-condition treatments, provisions is made to allow the programmer to supply a subroutine where some integration variables can be modified before the updating step.

The algorithm has been tested on artificially-generated data with good results. It has also been applied on a flow through a converging-diverging duct computed by Darmofal [11]. In this flow a separation bubble exists immediately downstream of the duct constriction. The tool is able to show a region of high residence time corresponding to the separation bubble. Applicability on multi-partitioned data has been demonstrated using a stator/rotor flow data computed by Saxer [34]. The stator and rotor data and processing are handled by separate pV3 clients. This example also illustrates the flexibility of the tool in handling special boundary cases such as the one presented by the stator/rotor interface.

## 5.2 Suggestions for Future Work

The line segments found by the swirl-flow finder can be colored to communicate other types of information about the flow. Further investigations must be done to determine what information are relevant and useful.

The residence-time algorithm can be adapted for tetrahedral cells. This improvement will allow the tool to work on all type of cells since (as described in chapter 3) other cells can be reduced to tetrahedra.

Further studies should be performed on the residence-time tool using various types of flows to determine other useful information that might be conveyed by this tool as well as to determine its limitations.

The results shown in appendix B suggests possibilities of using the eigensystem of the velocity-gradient tensor in finding shear surfaces. This potential use should be investigated further.

## Bibliography

- [1] R. H. Abraham and C. D. Shaw, *Dynamics: The Geometry of Behavior*, parts 1-4, Ariel Press, Santa Cruz, CA., 1984.
- [2] G. V. Bancroft, F. J. Merrit, T. C. Plessel, P. G. Kelaita, R. K. McCabe, and A. Globus, "FAST: A Multi-processing Environment for Visualization of Computational Fluid Dynamics," *Proceedings Visualization '90*, San Francisco, CA, Oct. 1990.
- [3] D. C. Banks and B. A. Singer, "A Predictor-Corrector Scheme for Vortex Identification," ICASE Report NO. 94-11, NASA CR-194882, 1994.
- [4] D. C. Banks and B. A. Singer, "Vortex Tubes in Turbulent Flows: Identification, Representation, Reconstruction," ICASE Report No. 94-22, NASA CR-194900, 1994.
- [5] S. Bryson and C. Levit, "The Virtual Windtunnel: An Environment for the Exploration of Three-Dimensional Unsteady Flows," *Proceedings Visualization '91*, October 1991.
- [6] M.. S. Chong, A. E. Perry, and B. J. Cantwell, "A General Classification of Three-Dimensional Flow Fields," *Phys. Fluids A*, Vol. 2, pp. 765-777, May 1990.
- [7] I. Currington and M. Coutant, "AVS - A Flexible Interactive Distributed Environment for Scientific Visualization Applications," Second Eurographics Workshop on Visualization in Scientific Computing, April 1991.
- [8] D. Darmofal. *Hierarchical Visualization of Three-Dimensional Vortical Flow Calculations*. Master's Thesis, M.I.T., March 1991.
- [9] D. Darmofal and R. Haimes, "Visualization of 3-D Vector Fields: Variations on a Stream," AIAA Paper 92-0074, 1992.
- [10] Darmofal and R. Haimes, "An Analysis of 3-D Particle Path Integration Algorithms for Unsteady Data," AIAA Paper 95-1713, 1995.
- [11] D. Darmofal. *A Study of the Mechanisms of Axisymmetric Vortex Breakdown*. Doctoral Thesis, M.I.T, November 1993.

- [12] M. Giles, personal communications.
- [13] M. B. Giles and R. Haimes, "Advanced Interactive Visualization for CFD," *Computing Systems in Engineering*, 1(1):51-62, 1990.
- [14] A. Globus, C. Levit, and T. Lasinski, "A Tool for Visualizing the Topology of Three-Dimensional Vector Fields," Report RNR-91-017, NAS Applied Research Office, NASA Ames Research Center, 1991.
- [15] R. Haimes, personal communications.
- [16] R. Haimes and D. Darmofal, "Visualization in Computational Fluid Dynamics: A Case Study," *Proceedings Visualization '91*, October 1991.
- [17] R. Haimes and M. Giles, "VISUAL3: Interactive Unsteady Unstructured 3D Visualization," AIAA Paper 91-0794, 1991.
- [18] R. Haimes, M. Giles, D. Darmofal, "Visual3 - A Software Environment for Flow Visualization," VKI Lecture Series on Computer Graphics and Flow Visualization in CFD, 1991.
- [19] R. Haimes, "pV3: A Distributed System for Large-Scale Unsteady CFD Visualization," AIAA Paper 94-0321, 1994.
- [20] R. Haimes, "Unsteady Visualization of Grand Challenge Size CFD Problems: Traditional Post-Processing vs. Co-Processing," *Proceedings of the ICASE/LaRC and ACM SIGGRAPH Symposium on Visualizing Time-Varying Data*, 1995.
- [21] M. G. Hall, "Cell-Vertex Multigrid Schemes for Solution of the Euler Equations," Technical Report 2029, Royal Aircraft Establishment, March 1985.
- [22] J. Helman and L. Hesselink, "Analysis and Representation of Complex Structures in Separated Flows," *SPIE Proceedings*, Vol 1459, 1991.
- [23] J. Helman and L. Hesselink, "Visualizing Vector Field Topology in Fluid Flows," *IEEE Computer Graphics and Applications*, May 1991.

- [24] D. G. Holmes and S. D. Connell, "Solution of the 2-D Navier-Stokes Equations on Unstructured Adaptive Grids," AIAA Paper 89-1932-CP, 1989.
- [25] J. Jeong and F. Hussain, "On the Identification of a Vortex," *Journal of Fluid Mechanics*, 285, pp. 69, 1995.
- [26] D. Jespersen and C. Levit, "Numerical Simulation of Flow Past a Tapered Cylinder," AIAA Paper 91-0751, 1991.
- [27] J. Kim and P. Moin, "The Structure of the Vorticity Field in Turbulent Channel Flow. Part 2. Study of Ensemble-Averaged Fields," *Journal of Fluid Mechanics*, 162, pp. 339, 1986.
- [28] P. Moin and J. Kim, "The Structure of the Vorticity Field in Turbulent Channel Flow. Part 1. Analysis of Instantaneous Fields and Statistical Correlations," *Journal of Fluid Mechanics*, 155, pp. 441, 1985.
- [29] R.-H. Ni, "A Multiple Grid Scheme for Solving the Euler Equations," *AIAA Journal*, 20, pp. 1565-1571, November 1981.
- [30] R.-H. Ni and J. C. Bogoian, "Prediction of 3-D Multi-Stage Turbine Flow Field Using a Multiple-Grid Euler Solver," AIAA Paper 89-0203, 1989.
- [31] A. E. Perry and H. Hornung, "Some Aspects of Three-Dimensional Separation, Part II: Vortex Skeletons," *A. Flugwiss, Weltraumforsch.* 8, Heft 3, pp. 155-160, 1984.
- [32] Rasure and Young, "An Open Environment for Image Processing Software Development," 1992 SPIE/IS&T Symposium on Electronic Imaging, *SPIE Proceedings*, Vol. 1659, February 1992.
- [33] Rasure and Kubica, "The Khoros Application Development Environment," *Experimental Environments for Computer Vision and Image Processing*, editor H.I. Christensen and J.L. Crowley, World Scientific 1994.
- [34] A. Saxer. *A Numerical Analysis of 3-D Inviscid Stator/Rotor Interactions Using Non-Reflecting Boundary Conditions*. Doctoral Thesis, M.I.T., February 1992.

- [35] R. A. Shapiro, "Prediction of Dispersive Errors in Numerical Solutions of the Euler Equations," MIT Computational Fluid Dynamics Laboratory Paper CFDL-TR-88-4, 1988.
- [36] C. Upson, T. Faulhaber, Jr., D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. van Dam, "The Application Visualization System: A Computational Environment for Scientific Visualization," *IEEE Computer Graphics and Applications*, Vol. 9, No. 4, pp. 30 - 42, July 1989.
- [37] S. A. Vermeersch. *Investigation of the F117A Vortical Flow Characteristics*. Master's Thesis, MIT, May 1993.
- [38] H. Vollmers, H. P. Kreplin, H. U. Meier, "Separation and Vortical-Type Flow around a Prolate Spheroid. Evaluation of Relevant Parameters," *AGARD Conference Proceedings*, No. 342.
- [39] L. A. Yates and G. T. Chapman, "Streamlines, Vorticity, Lines, and Vortices," AIAA Paper 91-0731, 1991.



## Appendix A

### Formulation of Lax-Wendroff Algorithm

The formulation of the Lax-Wendroff algorithm starts from the second-order Taylor series expansion of  $U^{n+1} = U((n+1)\Delta t)$ , where the superscript  $n$  denotes the time level. Thus,

$$U^{n+1} = U^n + \Delta t \left( \frac{\partial U}{\partial t} \right)^n + \frac{1}{2} \Delta t^2 \left( \frac{\partial^2 U}{\partial t^2} \right)^n \quad (\text{A.1})$$

Substituting Eqn. (4.9) into (A.1) and rearranging,

$$U^{n+1} = U^n - \Delta t \left( \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} + \frac{\partial H}{\partial z} - Q \right)^n - \frac{\Delta t}{2} \left( \frac{\partial}{\partial x} \Delta F^n + \frac{\partial}{\partial y} \Delta G^n + \frac{\partial}{\partial z} \Delta H^n - \Delta Q^n \right) \quad (\text{A.2})$$

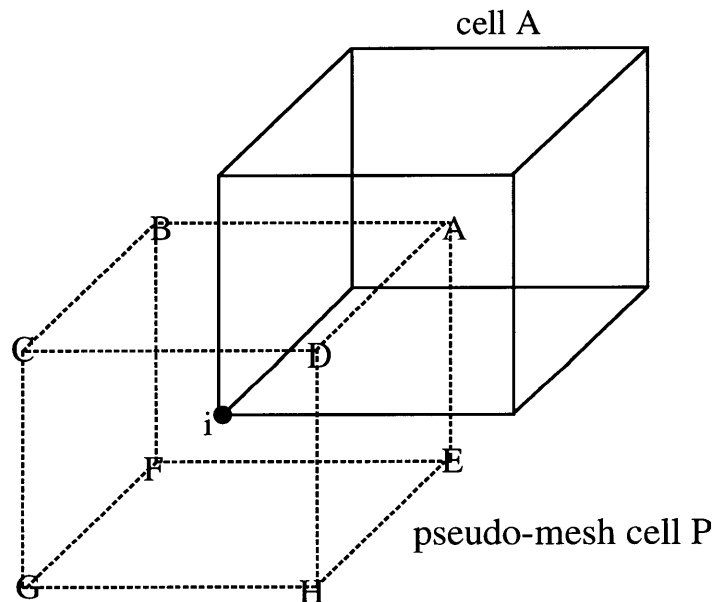


Figure A.1 Pseudo-mesh cell P (dashed) surrounding node  $i$  and a cell neighbor A. The corners of P are composed of the center of neighboring cells marked by capital letters.

where

$$\Delta F^n = \Delta t \frac{\partial F}{\partial t}, \Delta G^n = \Delta t \frac{\partial G}{\partial t}, \Delta H^n = \Delta t \frac{\partial H}{\partial t}, \Delta Q^n = \Delta t \frac{\partial Q}{\partial t}$$

Now consider the construction of the Lax-Wendroff algorithm on an unstructured grid, as illustrated in Figs. 4.1 and 4.2. Node  $i$  is surrounded by its eight nearest neighbor cells A, B, C, D, E, F, G, and H. The change at node  $i$  is  $\delta U_i \equiv U_i^{n+1} - U_i^n$ , which is found by integrating Eqn. (4.2) over the cell P of volume  $V_i$ , and then using Gauss's theorem.

$$\delta U_i = \frac{1}{V_i} (-\Delta t \iint_{cell P} (F, G, H)(n_x, n_y, n_z) dS + \Delta t V_i Q_i - \frac{\Delta t}{2} \iint_{cell P} (\Delta F, \Delta G, \Delta H)(n_x, n_y, n_z) dS + \frac{\Delta t}{2} V_i \Delta Q_i) \quad (A.3)$$

where  $\vec{n} = (n_x, n_y, n_z)$  is the unit normal vector pointing outward of the surface element  $dS$ .

$(\Delta F, \Delta G, \Delta H)(n_x, n_y, n_z)$  indicates the dot product of the two vectors.

Each of the integrals are considered separately. The first integral is the first-order flux contribution. It is approximated as the sum of one-eighth of the surface integral over the cells

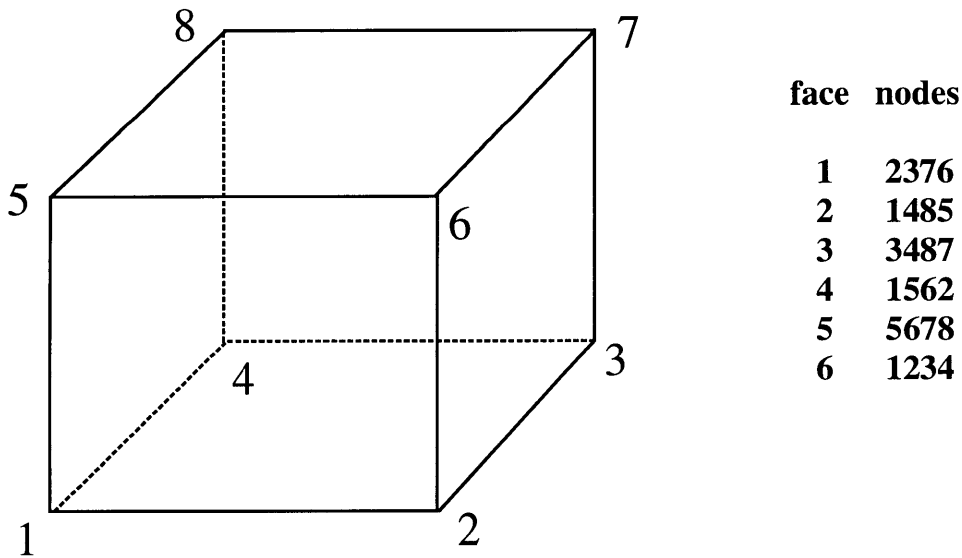


Figure A.2 Face and node numbering for hexahedral cell.

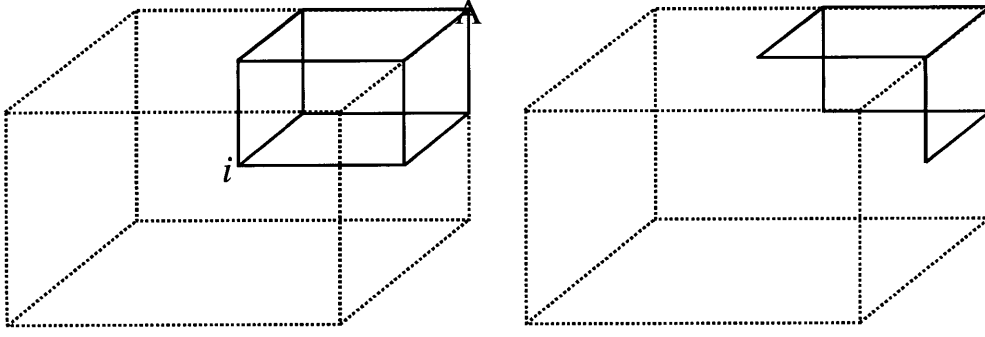


Figure A.3 Contributions of cell A to the first-order (left), and the second-order (right) flux integration.

A,..., H. The first order source term is approximated in a similar manner, as the sum of one-eighth of the average over each of the cells A, ..., H. The second integral, representing the second-order flux, is approximated as a sum of open surface integrals, as shown in Fig. A.3. The first and second order contributions of cell A to node  $i$  can be written as

$$\begin{aligned} \delta U_{iA} = & \frac{1}{V_i} \left( -\frac{\Delta t}{8} \iint_{cell A} (F, G, H)(n_x, n_y, n_z) dS + \frac{\Delta t}{8} V_A Q_A \right. \\ & - \frac{\Delta t}{2 \cdot 4} \iint_{faces (1-3-5)_A} (\Delta F, \Delta G, \Delta H)(n_x, n_y, n_z) dS \\ & \left. + \frac{\Delta t}{2 \cdot 8} V_A \Delta Q_A \right) \end{aligned} \quad (A.4)$$

Eqn. (A.4) is approximated by

$$\delta U_{iA} = \frac{1}{8} \left( \frac{\Delta t}{V} \right)_i \left\{ \left( \frac{\Delta t}{V} \right)_A \Delta U_A - \sum_{f=1,3,5} (\Delta F_A \bar{S}_x + \Delta G_A \bar{S}_y + \Delta H_A \bar{S}_z)_f + \frac{V_A}{2} \Delta Q_A \right\} \quad (A.5)$$

where  $\Delta U_A$  denotes the average first-order change of U in cell A, and is given by

$$\begin{aligned} \Delta U_A = & - \left( \frac{\Delta t}{V} \right)_A \iint_{cell A} (F, G, H)(n_x, n_y, n_z) dS + \frac{\Delta t}{8} V_A Q_A \\ = & - \left( \frac{\Delta t}{V} \right)_A \left( \sum_{f=1}^{6 \text{ faces}} (\bar{F} S_x + \bar{G} S_y + \bar{H} S_z)_f \right)_{cell A} + \Delta t Q_A + O(t^4) \end{aligned}$$

where the overbar means an average over the four nodes defining the face  $f$ .  $S_x$ ,  $S_y$ , and  $S_z$  are the projected areas of face  $f$  on the  $yz$ ,  $xz$ , and  $xy$  planes.  $\bar{S}_x$ ,  $\bar{S}_y$ , and  $\bar{S}_z$  are the averaged projection areas of opposite faces. The second-order flux terms is formulated in this way so that

the sum of their contributions to the corner nodes of a cell is zero, and thus preserving conservation. The evaluation of the terms  $\Delta F_A$ ,  $\Delta G_A$ ,  $\Delta H_A$ , and  $\Delta Q_A$  depend on the type of flow being considered, as shown in section 4.2.

The contributions from the other cells B,...,H to node i are formulated similarly. Finally, the residual  $\partial U_i$  can be expressed as

$$\delta U_i = \sum_{j=1}^{8 \text{ cells}} \delta U_{ij} = \delta U_{iA} + \delta U_{iB} + \delta U_{iC} + \delta U_{iD} + \delta U_{iE} + \delta U_{iF} + \delta U_{iG} + \delta U_{iH}$$

and the new value of U at node i is

$$U^{n+1} = U^n + \delta U_i$$

## Appendix B

# Identification of Flow Separation Using the Eigensystem of the Velocity-Gradient Tensor

The fundamental proposal is that flow separation can be identified using the eigenvalues and eigenvectors of the velocity-gradient tensor,  $\partial u_i / \partial x_j$ . As discussed in chapter 3, the flow pattern about a critical point (or about any point when viewed in a frame of reference moving with the point) is defined by the eigensystem of this tensor. An observation of the flow patterns, as illustrated in Fig. B.1, suggests that the eigenvector corresponding to the largest positive real eigenvalue ( $\lambda_{R+}$ ) points in the direction normal to the separation plane.  $\lambda_{R+}$  and  $\vec{X}_{R+}$  will be used

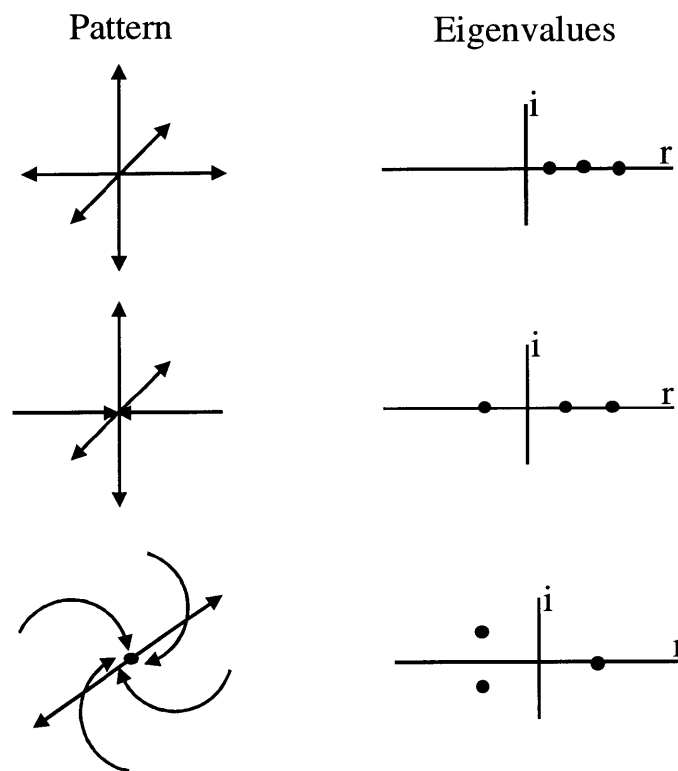


Figure B.1 Flow patterns for selected distributions of eigenvalues of the velocity-gradient tensor.

to denote this eigenvalue-eigenvector pair. Thus, the separation surface is a surface where the velocity vector is perpendicular to the local  $\vec{X}_{R+}$ .

The implementation of this algorithm employs an approach similar to that used for finding the center of swirling flow (see chapter 3). The scheme works on tetrahedra cells, and checks for the existence of a separation surface one cell at a time. The outline of the procedure is as follows:

1. Linearly interpolate the velocity within the cell to obtain

$$\mathbf{u}_i = \mathbf{a}_i + b_i \mathbf{x} + c_i \mathbf{y} + d_i \mathbf{z} \quad (\text{B.1})$$

2. Compute the velocity gradient tensor A. Since a linear interpolation of the velocity within the cell can be written as

$$u_i = C_i + \frac{\partial u_i}{\partial x} \Delta x + \frac{\partial u_i}{\partial y} \Delta y + \frac{\partial u_i}{\partial z} \Delta z \quad (\text{B.2})$$

then A can be constructed from the coefficients of the linear interpolant.

3. Find the eigenvalues and eigenvectors of A, and identify the eigenvector ( $\vec{X}_{R+}$ ) corresponding the largest positive real eigenvalue. If no positive real eigenvalue exists, continue with the next tetrahedral cell and go back to step 1.
4. First, let  $\vec{X}_{R+} = (x_1, x_2, x_3)$  and  $\vec{u} = (u_1, u_2, u_3)$ . Then, the plane where the velocity vector is everywhere normal to  $\vec{X}_{R+}$  can be found by setting the dot product of  $\vec{X}_{R+}$  and  $\vec{u}$  to zero.

$$\vec{X}_{R+} \cdot \vec{u} = x_1 u_1 + x_2 u_2 + x_3 u_3 = 0$$

↓

$$\begin{aligned} & (b_1 x_1 + b_2 x_2 + b_3 x_3) x + (c_1 x_1 + c_2 x_2 + c_3 x_3) y + (d_1 x_1 + d_2 x_2 + d_3 x_3) z \\ & + (a_1 x_1 + a_2 x_2 + a_3 x_3) = 0 \end{aligned} \quad (\text{B.3})$$

Note that Eqn. (B.3) is the equation of a plane.

5. The next step is to determine whether this plane intersect the cell being considered. The algorithm used to do this is already available within the pV3 system library, and will not be described here. The intersection (if any) defines the separation surface within the current cell.

This method has been tested on a number of artificially-generated data in which there is a distinct separation surface. Figs. B.2a to B.2c show the results on a data set in which the flow diverges at a 3-degree angle from the specified separation plane (shown as a dotted line in Fig. B.2a). The separation surface computed by the algorithm correspond well to the defined plane, as shown in Fig. B.2b and B.2c. The “stepping” effect is due to the discretization of the data. In Figs. B.3a to B.3c the flow moves in opposite direction across the separation plane. The location of separation plane is same as in the first case. Again, a good agreement is found between the result and the specified plane.

Although good results are evident in cases where there are clear separation surfaces and minimal amount of noise in the data, the result on actual CFD data with more complex flow features are less reliable. The algorithm has been tested on data for a swirling flow through a converging-diverging duct (see Fig. B.4a.) computed by Darmofal [11] using the incompressible, axisymmetric Navier-Stokes equations. In this result, a separation bubble (which correspond to a streamfunction value of 0) exists just downstream of the converging section, shown in Fig. B.4a as an iso-surface of streamfunction with value 0. Figs. B.4b and B.4c show the surfaces found by the algorithm. By overlapping the streamfunction iso-surface and the “separation” surfaces (see Fig. B.4d), it can be seen that only a small portion of the separation bubble is detected and that the result is corrupted by noise. It has been determined that the considerable amount of swirl in this flow results in a flow with a predominantly spiral-saddle pattern (i.e., velocity-gradient tensor with 1 real and a pair of complex-conjugate eigenvalues) with an axis of rotation approximately parallel to the duct axis. Consequently, only portions at the front and back of the separation bubble are detected because only in those regions of the bubble are the flow approximately perpendicular to the axis.

Therefore, in view of its limitations, this method is deemed unreliable and to have limited practical usefulness for identifying flow separations.

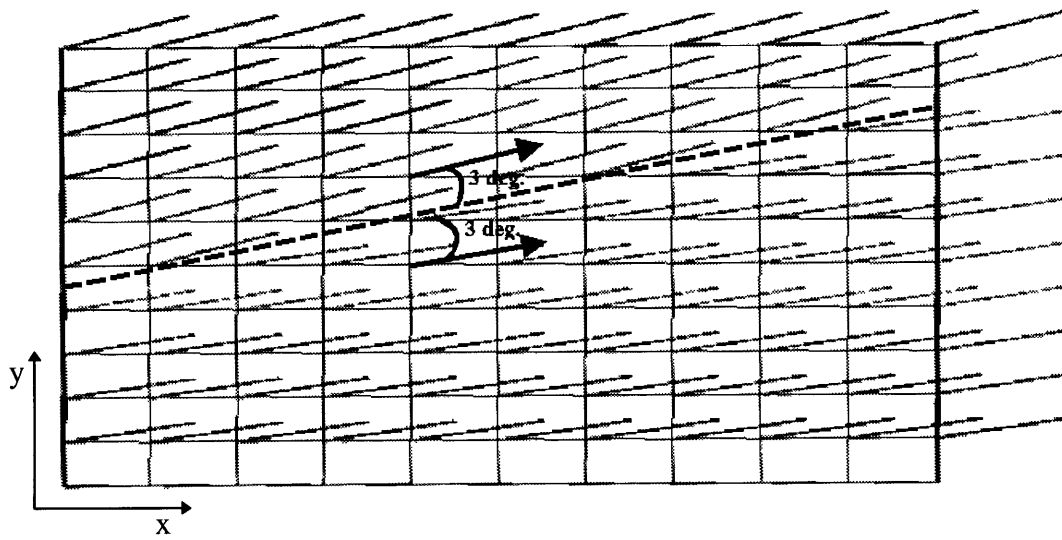


Figure B.2a Flow separating at a 3-degree angle from the specified separation plane (dotted line). Velocity is zero and uniform in the  $z$  direction.

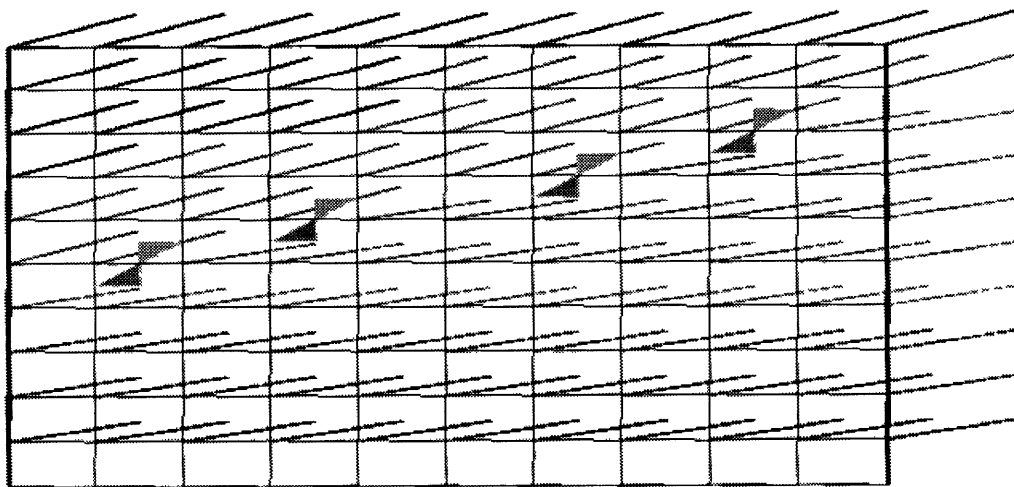


Figure B.2b Side view of separating flow and the surface found by the separation-flow-finder algorithm.



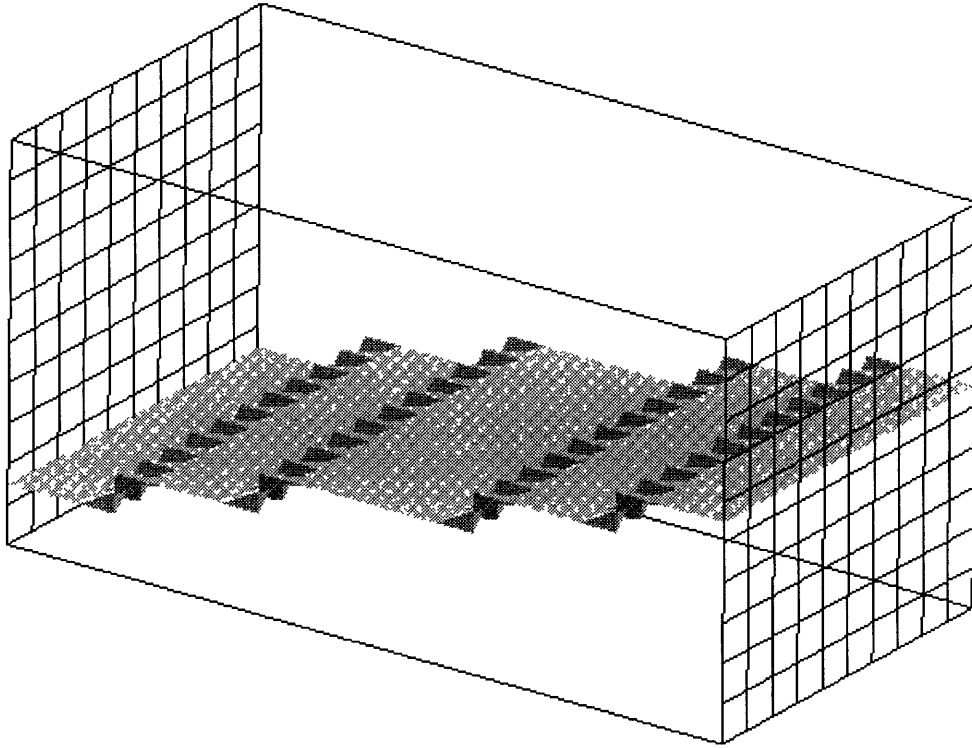


Figure B.2c Separation surface shown in Fig. B.2b. Velocity vector cloud not shown.

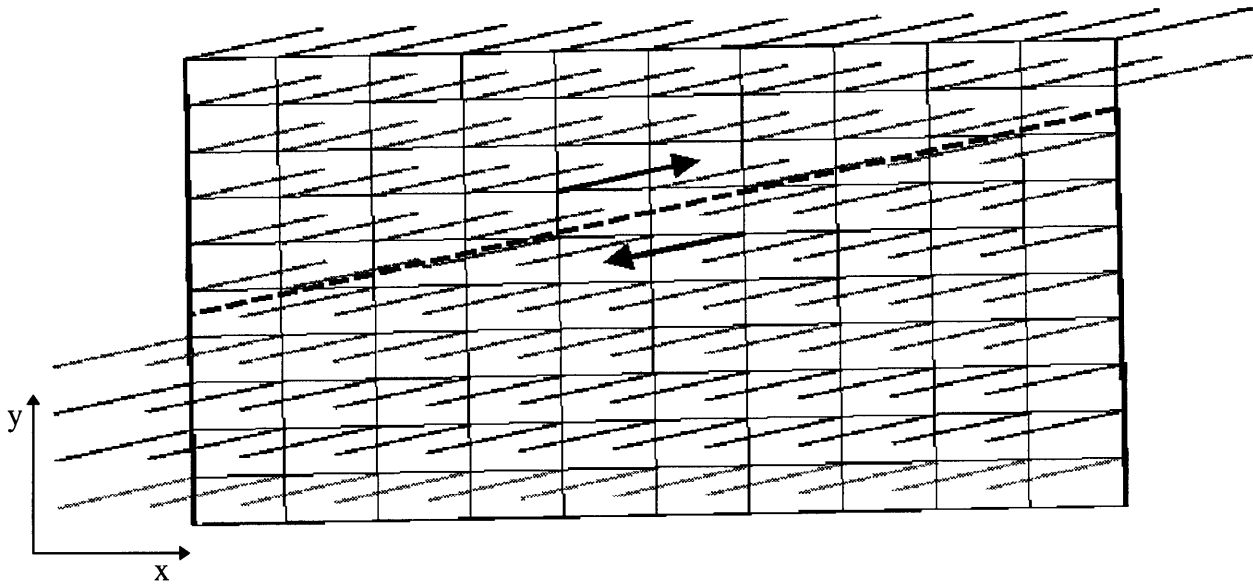


Figure B.3a Flow moving in opposite direction divided by a separation plane (dotted line). Velocity in is zero and uniform in the  $z$  direction.

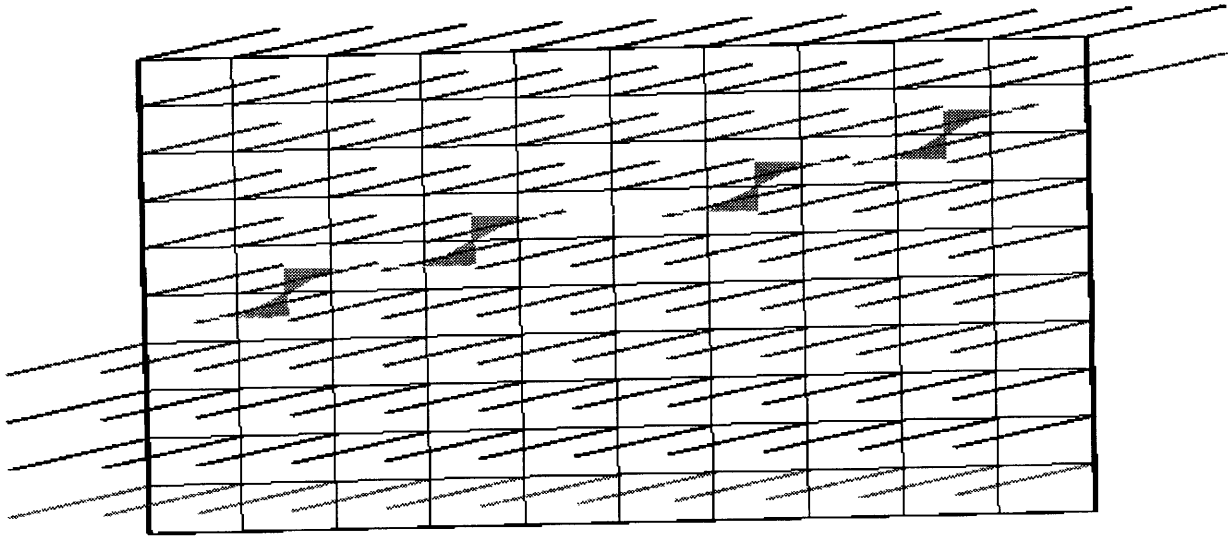


Figure B.3b Side view of opposing flow and the surface found by the separation-flow-finder algorithm.

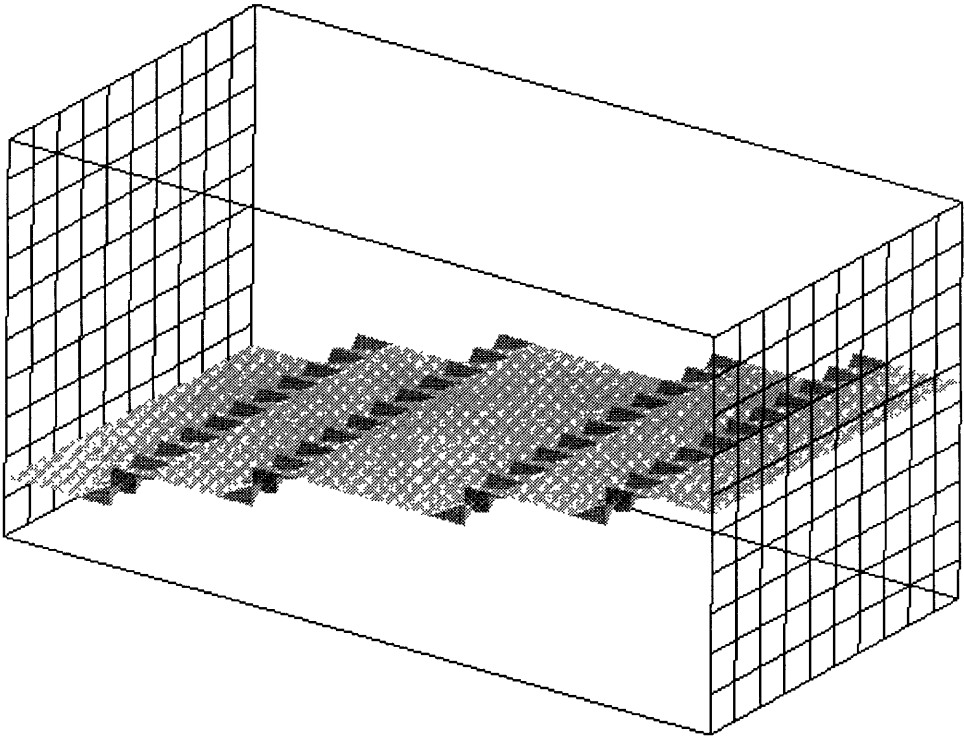


Figure B.3c Separation surface shown in Fig. B.3b. Velocity vector cloud not shown.

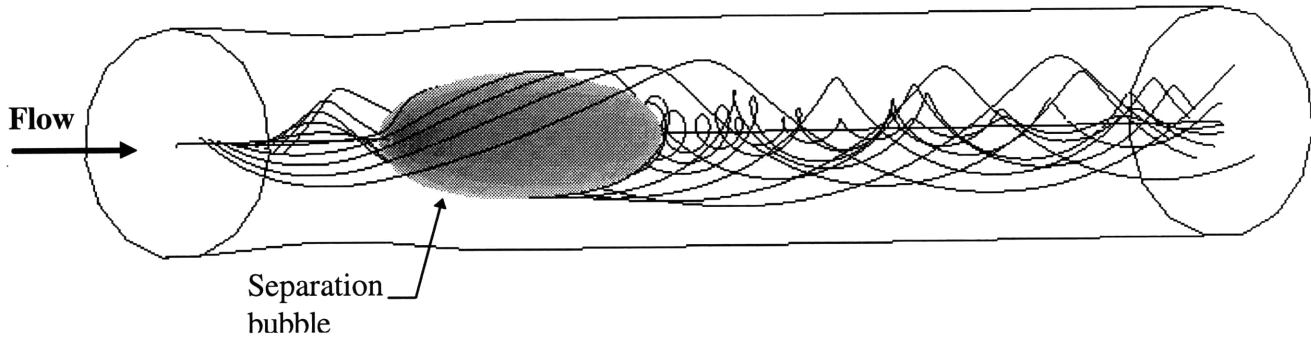


Figure B.4a Swirling flow through a converging-diverging duct [11]. A separation bubble is shown as an iso-surface of streamfunction with value 0. Streamlines indicate strongly swirling flow with axis of rotation approximately parallel to the duct axis.

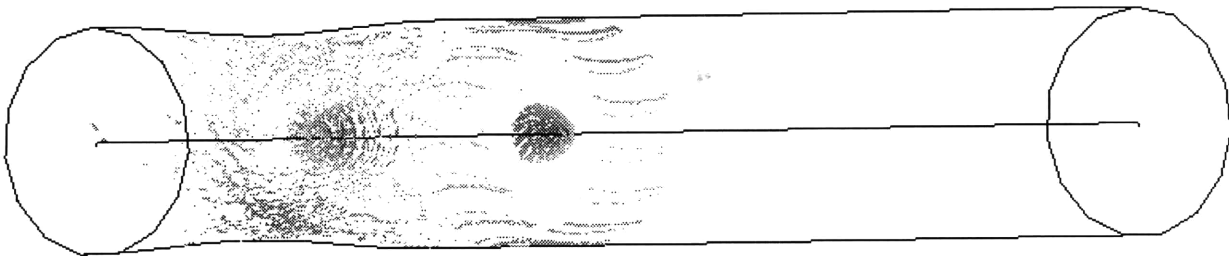


Figure B.4b Surfaces found by the separation-flow-finder algorithm.

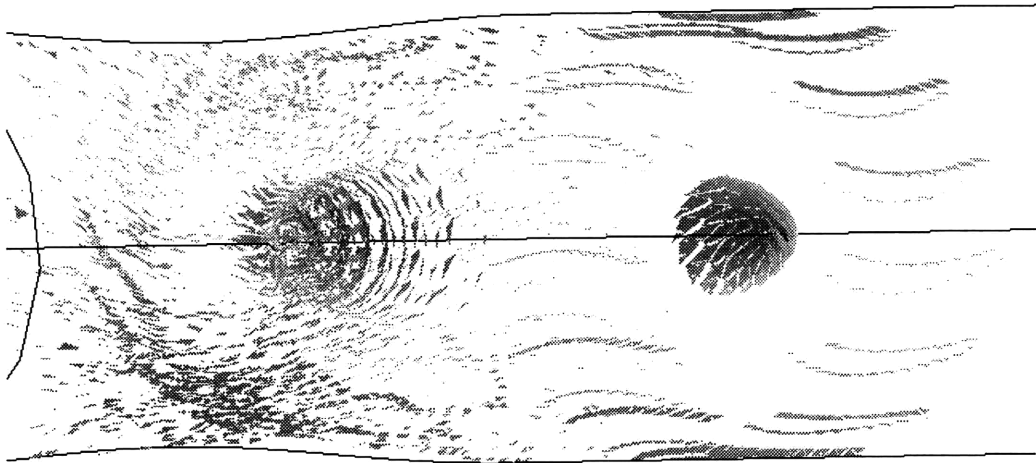


Figure B.4c Enlargement of Fig. B.4b.

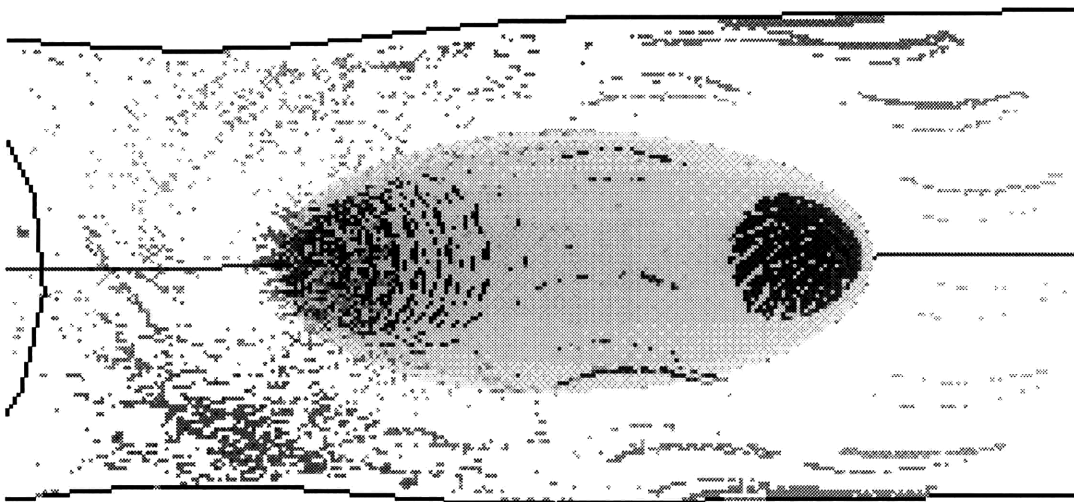


Figure B.4d Separation surfaces and streamfunction iso-surface overlapped.