Stellingen

behorende bij het proefschrift

Extraction and Visualization of Geometries in Fluid Flow Fields

Ari Sadarjoen

12 april 1999

1. Een standaard representatie van topologische informatie in ongestructureerde roosters is wenselijk voor visualisatie-algoritmen.

2. Geometrische criteria voor het detecteren van wervels zijn principieel beter dan fysische criteria.

3. Het vinden van zwakke wervels is een zwak punt van de meeste technieken voor werveldetectie.

4. Voor visualisatie van een data-veld moet een deformeerbaar oppervlak geen interne vervormingsweerstand hebben, maar zich volledig voegen naar het veld.

5. Bij kunst is er dikwijls spanning tussen vorm en inhoud, bij visualisatie is het de kunst om vorm aan de inhoud te geven.

6. Voor het verkrijgen van nuttige informatie zijn bibliotheken nog steeds doelmatiger dan het internet.

7. Intuïtieve systemen zouden geen dikke handleidingen moeten hebben.

8. Mensen die Nederlands willen leren worden niet aangemoedigd door de neiging van Nederlanders om alles te spreken behalve Nederlands.

9. Spellingshervormingen leiden eerder tot meer uitspraakfouten dan tot minder spelfouten.

10. Van de zon kan men het beste in de schaduw genieten.

1. A standard representation of topological information in unstructured grids is desirable for visualization algorithms.

2. Geometric criteria for vortex detection are in principle better than physical criteria.

3. Detecting weak vortices is a weakness of most vortex detection methods.

4. For visualization of a data field, a deformable surface should not have any internal smoothing energy, but it should completely adapt to the field.

5. In art, there is often tension between form and content; in visualization, it is an art to show content through form.

6. For obtaining useful information, libraries are still more effective than the Internet.

7. Intuitive systems should not require bulky manuals.

8. People wishing to learn Dutch are not encouraged by the tendency of the Dutch to speak anything but Dutch.

9. Spelling reforms result in more pronunciation errors rather than fewer spelling errors.

10. The sun is best enjoyed in the shade.

Extraction and Visualization of Geometries in Fluid Flow Fields

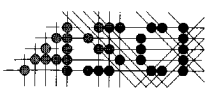
About the cover background
An embossed and rotated version of Colour Plate 6, which visualizes the Bay of Gdańsk (Poland) using streamlines and vortices approximated by ellipses (see Section 6.2). Data courtesy WL | Delft Hydraulics.

About the front cover
A visualization of flow patterns in the harbour of Lith (The Netherlands) and the Maas river, using particle paths coloured with the velocity magnitude (see Section 3.5.2). The colouring scheme uses a rainbow scale, with red showing the highest values and blue the lowest values. The harbour (on the right) is separated from the river (on the left) by a thin dam, and contains a recirculation zone with much lower velocities than in the river. Data courtesy WL | Delft Hydraulics.

About the back cover
Top image: transitional pipe flow, with vortices visualized by yellow ellipsoid icons and selective streamlines (see Section 6.3.4). Red shows streamlines in the left half of the pipe, white shows streamlines in the right half. Part of the cylinder wall with gridlines is shown in dark blue to increase the sense of depth. Data courtesy Lab. for Aero- and Hydrodynamics of Delft University of Technology.

Middle image: flow past a tapered cylinder, with global streamlines in yellow and vortices approximated by ellipses (see Section 4.5.2). The red ellipse indicates counter-clockwise rotation, the green ellipse clockwise rotation. The background is a grid slice coloured (using the rainbow scale) with the λ_2 scalar quantity, low values of which are supposed to indicate vortices. Data courtesy NASA Ames Research Center.

Bottom image: flow in the Pacific Ocean, with the continent of North America shown in red. The figure shows streamlines coloured with the velocity magnitude (using the rainbow scale) and white curvature centre density peaks indicating vortices (see Section 4.3.2).



This work was carried out in graduate school ASCI.
ASCI dissertation series number 43.

Extraction and Visualization of Geometries in Fluid Flow Fields

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft
op gezag van de Rector Magnificus prof.ir. K.F. Wakker
in het openbaar te verdedigen ten overstaan van een commissie,
door het College voor Promoties aangewezen,
op maandag 12 april 1999 om 13:30 uur
door

Ignatius Armatessa SADARJOEN

informatica ingenieur
geboren te Mainz (Duitsland)

Dit proefschrift is goedgekeurd door de promotor:
Prof.dr.ir. F.W. Jansen

Samenstelling promotiecommissie:
Rector Magnificus, voorzitter
Prof.dr.ir. F.W. Jansen, Technische Universiteit Delft, promotor
Prof.dr.ir. C.J. van Duijn, Technische Universiteit Delft
Prof.dr.ir. J. Biemond, Technische Universiteit Delft
Prof.dr.ir. A.E. Mynett, IHE / WL Delft Hydraulics
Prof.dr.ir. C.A. Grimbergen, Universiteit van Amsterdam
Prof.dr. M. Rumpf, Universität Bonn

Ir. F.H. Post heeft als begeleider in belangrijke mate aan de totstandkoming van het proefschrift bijgedragen.

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Sadarjoen, Ignatius Armatessa

Extraction and Visualization of Geometries in Fluid Flow Fields / Ignatius Armatessa Sadarjoen . - [S.l. : s.n.]. Ill.
Thesis Technische Universiteit Delft. - With ref. - With summary in Dutch.
ISBN 90-6464012-2
NUGI 855
Subject heading: scientific visualization / computer graphics

Preface

The research described in this thesis was conducted at the Computer Graphics and CADCAM group of the Faculty of Information Technology and Systems of Delft University of Technology. It is the fourth in a series of PhD projects concerned with scientific visualization. The first to receive a PhD degree in visualization was Andrea Hin, who developed visualization techniques for turbulent flow [Hin, 1994]. A year later, Theo van Walsum graduated on the subject of selective visualization on curvilinear grids [van Walsum, 1995]. Next, Wim de Leeuw's thesis described methods for presentation and exploration of flow data [de Leeuw, 1997]. All of these projects were concerned with visualization of fluid flows. My project continues along this line, although the focus is different: it concentrates on flow visualization using geometries, a collective name for curves, surfaces, and volumes.

Many people have contributed to this dissertation, in many different ways and degrees, some of whom I would like to thank in particular. The first and foremost figure I wish to thank is my supervisor Frits Post, who has had the largest influence on my work. His close involvement in my project has proven invaluable. During our frequent meetings and discussions, he was always full of new ideas and suggestions for me to try out (or ignore). When roads seemed a dead end, he always managed to fill me with his enthusiasm to find a way out, or to continue in other directions. I also greatly appreciate his thorough reading of and commenting on all my manuscripts, including this thesis.

I thank my promotor Erik Jansen for offering me the opportunity to do this project in his group, for his enthusiastic support throughout the project, and in the final stage, for accurately yet quickly reading this manuscript.

I want to thank some people from WL | Delft Hydraulics, my co-supervisor Arthur Mynett, for his valuable time and comments and ideas provided during regular meetings, Jan Mooiman, for providing data sets and insights into their physical backgrounds, and Irving Elshoff, for his assistance in all matters related to AVS/Express and PLANKTON-97.

I am also grateful to my new employers at the Manchester Visualization Centre for patiently having waited for me to finish my thesis full-time in Delft rather than part-time in Manchester.

Two MSc students contributed to this PhD project. Ton van der Wouden designed and implemented `CNX-lib`, a well thought out library for unstructured grids.

Alex de Boer reincarnated the PLANKTON particle tracer into PLANKTON-97 using AVS/Express, with several significant improvements.

I would also like to thank Bing Ma from Tsinghua University in Beijing, who during his six-month stay in Delft provided me with data sets of the transient pipe flow and a wealth of background information.

I am happy to have worked with David Banks from Mississippi State University, and Hans-Georg Pagendarm from the German Aerospace Centre, in a long-distance collaboration which formed the basis for a large part of Chapter 4 on vortex detection.

The Knowledge Based Systems group of our Faculty kindly allowed me to use the Matlab package on their machines, which was indispensable for many of the 2D figures in this thesis.

My office mate Freek extended his feature viewer to be (ab)used by me as a particle renderer, and contributed to an enjoyable work atmosphere through daily discussions (and distractions) about work and many other things. My other colleagues in the group, Alex, Erik, Klaas Jan, Marco, Maurice, Michal, Paul, (A.) Rafa, Theo, and Winfried, also contributed to a pleasant work atmosphere, by regularly dropping into our office to show their exciting new results, and to guarantee the tea supply.

The system administrators Aadjan, Kees, Peter, Piter, and last but certainly not least, Ruud, did their utmost to provide all the necessary hardware and software facilities. Non-technical but no less important support was provided by our secretaries Toos and Coby, "The Mothers" of the group.

I am grateful to Mei-Ling Hsu, who looked up the Chinese originals of the poems decorating several chapter headings, and retyped them several times. Xièxiè! Evelien Rusch was so kind to look up the Swedish poems on very short notice, and type them in. Tack så mycket!

Erik Vullings, a PhD student in the Dept. of Electrical Engineering, had a significant influence on the course of my project, by providing me with useful technical and procedural information.

For the musical decoration during working hours, I would like to thank W.A. Mozart for composing such wonderful music, which always had a stimulating and refreshing effect, and which revitalized me whenever inspiration had run dry.

I thank Driejana, who kept on encouraging me to finish soon and who helped to design the cover.

Finally, I am grateful to my parents for all their love and unconditional support; without those, this work would not have been possible.

I. Ari Sadarjoen

Contents

Chapter 1

Introduction

Salt, bittersalt [1]
är havet, och klart och kallt.
På djupet multnar mycket,
men havet renar allt.

1.1 Scientific visualization

Scientific visualization could be described as the art of translating numbers into a clear visual representation, which makes it easier for people to interpret the numbers. Scientific visualization has a long history: an early well-known example dates back from 1861, when Charles Joseph Minard created an ingenious graphical representation of numerical multidimensional data concerning Napoleon's unsuccessful campaign to Russia in 1812–1813. This visualization is shown in Figure 1.1 [Tufte, 1990]. The chart visualizes the size of Napoleon's army which is dramatically reduced as the campaign proceeds through various locations. On the way to Moscow, indicated by the hatched band, the army size decreases from 422,000 to 100,000. On the way back, indicated by the solid black band, the army is literally decimated to 10,000. This graph visualizes six variables: the army position (longitude and latitude), its size, its direction of movement, the temperature, and the date.

Before the advent of computers, scientists who conducted experiments also developed ways of visualizing the results. Many of you may remember from the science classes in high school that magnetic field lines can be visualized by iron shavings, which are oriented when a magnet is put under a sheet of paper. A slightly more advanced way of visualization is the use of aluminium particles released in a flow and photographed at regular intervals.

In modern times, scientific research and engineering practice increasingly employ computer simulations and automatic measurements, which result in large amounts of numbers. Simulations are performed by scientists for modelling complex physical phenomena, about which they try to get a better understanding. Simulations are also performed by engineers who want to test and improve their designs, without having to build costly prototypes or scale models. Measurements are often taken in large amounts by satellites, which orbit the earth or are sent out to other planets in the solar system. These computer simulations and measurements produce large amounts of data, called data sets, typically in the form of numbers. As the power of computers has steadily increased over the years, this has allowed scientists and engineers to develop

[1] Salt, bitter salt // is the sea, and clear and cold // In the deep, much decays // but the sea cleans everything. (Karin Boye: *Havet*)

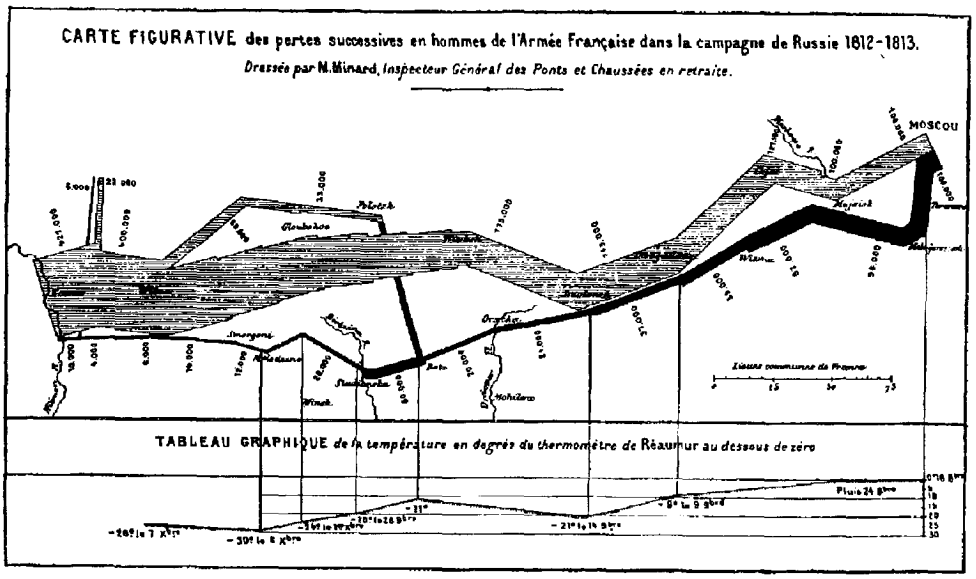


Figure 1.1: Minard's visualization of Napoleon's campaign to Russia [Tufte, 1990]

more complex and more accurate models that produce larger and larger data sets, which have to be visualized.

Therefore, scientific visualization has become necessary for scientists and engineers, in order to obtain insight in the results of their models and measurements. Images are much better suited for processing by the human visual system, hence the well-known proverb *"One picture says more than a thousand words"* (or numbers in this case).

1.2 Objectives

The research in this thesis describes the development of techniques to assist the scientific user in coping with the large amounts and high complexity of the data, by providing interactive techniques for exploring the data. What the user is usually interested in, are the characteristic phenomena of the data, so-called *features*. Examples of features are vortices, boundary layers, and shock waves. Through these features, the amount of data and its complexity can be reduced. Therefore, we have developed techniques which extract features from data sets and visualize them.

One classification for visualization techniques distinguishes between: global techniques, geometric techniques, and feature-based techniques [Post *et al.*, 1999]. *Global techniques* render data directly, without deriving curves or polygons first. Global techniques include arrow plots, direct volume rendering, and direct surface rendering. An

advanced global technique, which uses texture to visualize vector fields, is Spot Noise [de Leeuw, 1997]. *Geometric techniques* first extract *geometric objects*, such as curves, surfaces and solids, and then render them for visualization. Examples are streamlines and stream surfaces. *Feature-based techniques* first extract *features* and then visualize them with any of the previous techniques or with abstract iconic representations. Examples of the latter are given in [van Walsum *et al.*, 1996].

The focus in this thesis is on geometric techniques, although sometimes we employ elements from feature-based techniques as well. An important reason is that geometries are easy to perceive and easy to render. This leads to the main objective of this thesis:

to describe techniques for extracting and visualizing geometries in fluid flow fields.

The application area in this thesis is fluid flows, both hydrodynamic and aerodynamic, because this area contains a wealth of interesting problems, and because we had access to experts and resources in that area. Within this field, we concentrated on Computational Fluid Dynamics (CFD), the field which develops numerical models of fluid flows and performs flow simulations based upon those models.

1.3 Structure of this thesis

The remainder of this thesis has the following structure. Chapter 2 contains an overview of related work in geometric techniques, and serves as a framework for placing the techniques covered in the following chapters into context. Chapter 3 describes particle tracing, a technique for visualizing velocity fields. Chapter 4 covers techniques for detecting vortices. Chapter 5 investigates deformable surfaces, a class of geometric objects that can be used for extracting a range of features which may be well represented by surfaces. Chapter 6 presents some applications of the techniques to some larger, 'real-life' examples. Finally, Chapter 7 contains conclusions and gives directions for future work.

Chapter 2

Geometry extraction techniques

Bengawan Solo, ...[1]
Mata airmu dari Solo
terkurung gunung seribu
Air mengalir sampai jauh
akhirnya ke laut.

This chapter reviews related work concerning geometry extraction. The purposes of this chapter are to show the context of our work and to provide a basis for the discussions in the following chapters.

As stated in Chapter 1, it is possibly to classify visualization techniques as direct techniques, geometric techniques, and feature-based techniques [Post *et al.*, 1999]. The focus in this thesis is on geometric techniques, although sometimes we employ elements from feature-based techniques as well. Geometric techniques produce geometries, which have two important advantages:

- geometries are easy for people to interpret. The visual system of the human brain is well adapted to recognizing shapes and colour properties of geometries like surfaces and curves.
- geometries are easy for computers to display. Modern hardware systems of computers are specialized in rendering lines and surfaces with colours and textures, to create a clear representation.

Several important types of geometries are:

- curves
- surfaces
- solids
- deformable models

Curves are often used to get a global view of the flow field. Surfaces are often used to get a local view of specific surface features, such as stream surfaces or separation surfaces. Deformable models are a special type of curves and surfaces, which start from some initial shape, and grow or shrink towards some target shape of a feature. In addition, a vortex is not a type of geometry, but a type of feature in fluid flows, which may be represented by various types of geometries, such as curves and surfaces.

The remainder of this chapter is organized as follows. Section 2.1 gives definitions of several kinds of data fields and grids. Section 2.2 describes various flow curves,

[1]Solo River, ... // Your source is in Solo // confined by a thousand mountains // Your water flows quite far // and finally into the sea. (Indonesian traditional)

and Section 2.3 various flow surfaces. Section 2.4 covers deformable models. Finally, Section 2.5 describes techniques for vortex detection.

2.1 Fields and grids

2.1.1 Fields

A three-dimensional field can be represented analytically by a global function $f(x, y, z)$, defined over a bounded spatial domain in R^3 [Silver *et al.*, 1999]. A unique field value s at every point (x, y, z) of the domain can be found by evaluation: $s = f(x, y, z)$ everywhere in the domain. This is usually not the case with the discrete numerical fields that are more common in science and engineering, where data values are known only at a finite number of data points. Such discrete fields are usually generated by data acquisition systems or numerical computer simulations.

Measured data fields can be generated by medical imaging systems, such as computed tomography (CT), magnetic resonance imaging (MRI), or positron emission tomography (PET) scanners. Other sources of large measured data fields include remote sensing systems of satellites, scanning electron microscopes, and seismic and acoustic sensing for underwater observations. For simplicity, we will assume in the rest of this thesis that the data fields have been generated by numerical simulations.

Physical models often consist of partial differential equations that cannot be solved globally and analytically. Therefore, discrete methods such as finite elements, finite differences, or finite volume methods are often used to numerically solve local equations. These methods are based on defining a computational grid, consisting of nodes and cells. Approximated equations are specified, resulting in a system of equations that can be solved numerically at each grid node or cell.

The domain of a simulation may have two or three spatial dimensions. It may also be varying in time. The data points (or grid points) thus are 2D (x, y) or 3D (x, y, z) coordinate positions. The data fields may contain any combination of scalar quantities (e.g. pressure, density, or temperature), vector quantities (e.g. force or velocity), or tensor quantities (e.g. stress or deformation) at each data point. The data values may be constant in time, or vary as a function of time. Time-dependent fields are important for highly dynamic phenomena such as fluid flow.

2.1.2 Grids

There are many types of computational grids, depending on the simulation technique, the domain, and the application [Silver *et al.*, 1999]. A grid consists of nodes and cells. The nodes are points defined in the simulation domain, and the cells are simple spatial elements connecting the nodes: triangles or quadrangles in 2D, tetrahedra or hexahedra in 3D. The cells must fill the whole domain, but may not intersect or overlap, and adjacent cells must have common edges and faces. Grids can be classified according to

their geometry, their topology, and their cell shape. Three of the most important types are discussed below.

The simplest type is the *uniform grid*, also called regular orthogonal, or Cartesian grid (see Figure 2.1a). This type of grid has a regular geometry and topology: the nodes are spaced in a regular array, and the cells are all unit cubes. The grid lines connecting the nodes are straight and orthogonal. Every node can be referenced by integer indices (i, j, k). Adjacent nodes can be found by incrementing any of the index vector components. Many operations on this type of grid (such as searching the grid cell which contains a given point) are very simple, but grid resolution is constant throughout the domain, and the shape of the domain must be rectangular.

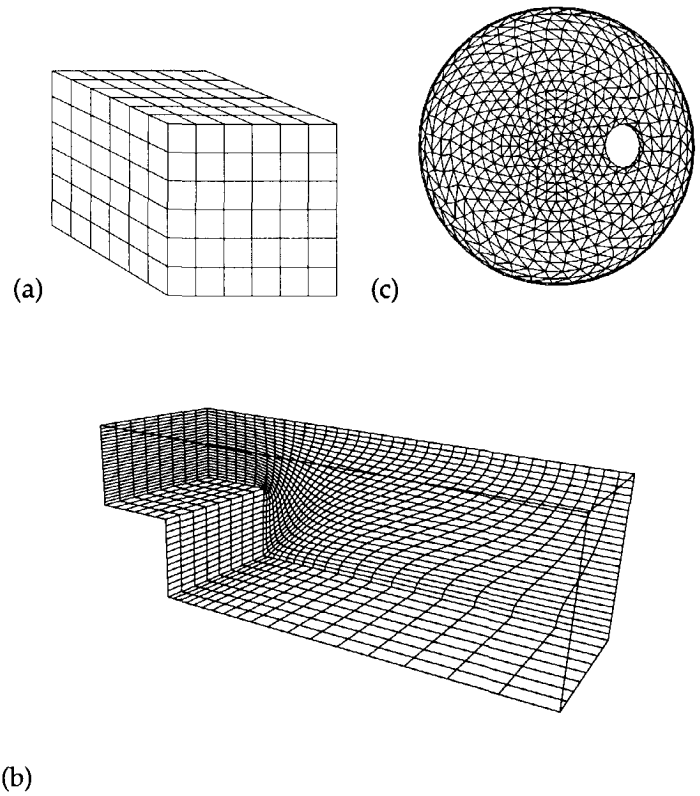


Figure 2.1: Several grid types: (a) regular orthogonal (Cartesian) grid, (b) structured curvilinear grid, and (c) unstructured grid

The second type of grid is the *structured curvilinear grid* (see Figure 2.1b). This type has a regular topology (the adjacency pattern for each internal node is the same), with the nodes again referenced by integer indices (i, j, k), and adjacent nodes can be found

by incrementing index values. The cells are usually hexahedra, with a deformed-brick shape. The geometry of each cell is irregular, and the cell faces are non-planar quadrangles. The cell size of a curvilinear grid can be highly variable, and thus the resolution of the simulation can be higher in areas of strong variation. Also, the curvilinear shape can be made to conform to the boundary of a curved object, such as an airplane wing. This type of grid is common in finite volume CFD simulations.

The third type of grid is the *unstructured grid*, where the topology and geometry are both irregular. Figure 2.1c shows an example from a finite element application [van den Broek *et al.*, 1998]. The nodes do not have a fixed adjacency pattern, and adjacency information cannot be derived from a spatial index, but has to be stored explicitly. The cells are usually triangles in 2D or tetrahedra in 3D. Cell size can be varied according to the amount of detail desired, and can be used to model a complex geometry. Unstructured grids are often used in finite element analysis. Due to the simple cell geometry, calculations on a single cell are simple.

There are many more variations of grids: staggered grids, hybrid (mixed-type) grids, multi-block grids, moving grids, and multi-resolution grids. In this thesis, we will concentrate mainly on the three types described before.

A numerical simulation will generally produce a discrete data field, consisting of a combination of scalar, vector, or tensor quantities, given at every grid node. These datasets can be very large, with as many as 10^4 to 10^6 nodes, and ten or more variables defined at every node. This can result in a size of up to hundreds of megabytes for stationary (time-independent) fields and up to terabytes for time-dependent fields.

2.2 Curves

Curves are usually easy to generate, easy to render, and easy to understand. An important subclass of curves are *flow curves*, which are used to visualize flow patterns. The definitions of several flow curves are listed below [Silver *et al.*, 1999].

- *streamline*: a tangent curve of a steady velocity field. Tangent curves are defined as curves that are everywhere tangent to the vector field. A streamline satisfies the equations: $dx/u = dy/v = dz/w$, where (u, v, w) are the velocity components in the x-, y-, and z-direction of the domain.
- *pathline*: a trajectory curve of a single fluid particle moving in the flow. This curve is identical to an integral curve obtained by stepwise integration of the velocity vector field (see Chapter 3 on the generation of pathlines).
- *streak line*: a line joining the positions at one instant of all particles that have been released from a single point over a given time interval $t_0 \ldots t_n$.
- *time line*: a line connecting all particles that have been simultaneously released in a flow from positions on a straight line, perpendicular to the flow direction. The straight line moves and deforms with the flow due to local velocity variations.
- *vorticity line*: a tangent curve of a vorticity vector field, which is the curl of velocity field: $\omega = \nabla \times \mathbf{v}$.

- *hyperstreamline*: a tangent curve of an eigenvector (usually with the largest magnitude) of a tensor field [Delmarcelle & Hesselink, 1993].

In a steady flow (also called stationary or time-independent flow), where the velocity field is constant in time, pathlines, streamlines, and streak lines are identical [Strid *et al.*, 1989]. In an unsteady flow (also called instationary or time-dependent flow), these curves may all be different. Streamlines, vorticity lines, and hyperstreamlines are mathematical abstractions, but they are all based on the idea of field lines. Many of these curves have been inspired by experimental visualization [Post & van Walsum, 1993].

Most of the curves are based on the notion of particle advection. They can be generated in a straightforward way, by integrating the vector field, which results in an integral curve. Particle tracing is a visualization method which simulates the release of massless particles, calculates their paths, and then renders them in an animation at a constant frame rate. Particle tracing is a common way of visualizing the results of CFD flow simulations; an extension for turbulent flow was introduced in [Hin & Post, 1993; Hin, 1994].

The choice of release points for particles or flow curves is critical. If flow curves are released too sparsely, or in the wrong regions, important flow features may be missed. If they are released too densely, cluttering may occur. An important technique for an even distribution of the lines is described in [Turk & Banks, 1996]. Recently, this technique was extended to curvilinear grids [Mao *et al.*, 1998].

Effectively rendering 3D curves is much more difficult than rendering 2D curves, due to occlusion, and the lack of direction information and depth cues. In [Fuhrmann & Gröller, 1998], some solutions are proposed, such as an algorithm for an even distribution of curves, and the use of texture for depth cues.

Our focus in this thesis is on the calculation of pathlines, or particle tracing in special grid types: in structured curvilinear grids (see Section 3.2), in σ-transformed grids, which are common in hydrodynamic applications (see Section 3.3), and in unstructured grids, which are more common in aerodynamics (see Section 3.4).

2.3 Surfaces

Curves are easy to visualize, but they do not carry much spatial information, and are therefore difficult to locate visually in space [Post & van Walsum, 1993]. Surfaces are much better for visualization, because lighting and shading are effective cues for perception of 3D shapes by a human observer.

Our focus in this thesis is on the generation of flow surfaces of three kinds. The first kind is adaptive isosurfaces, which can be used to approximate local isosurfaces with the desired accuracy specified by the user (see Section 5.5.1). The second kind is separation surfaces (see Section 5.5.2), which are used to find recirculation zones. The third kind is vortex tubes (see Section 6.4). For completeness, a brief introduction to isosurfaces, stream surfaces, and implicit surfaces is given below.

2.3.1 Isosurfaces

An isosurface is a collection of points (x, y, z) in a scalar field $f(x, y, z)$ which have the same field value: $f(x, y, z) = C$, where C is a constant value called the isovalue. These points are usually connected by a polygon surface. The classic algorithm to generate isosurfaces is the marching cubes algorithm described in [Lorensen & Cline, 1987]. Based on this algorithm, many improvements, optimizations, and variations have been produced, but these do not fall within the scope of this thesis.

While isosurfaces are certainly useful for extracting certain types of features from data sets, they also have drawbacks. Standard isosurface algorithms work with a global isovalue, which may lead to surface fragments throughout the entire data set. This may not always be desirable, when we are only interested in a certain region of interest. Another drawback is that standard algorithms generate large amounts of polygons, which has a strong relation to the number of grid cells. Sometimes, this is reduced afterwards with the use of surface simplification or decimation algorithms, at the cost of extra processing time.

A better option would be to prevent too many polygons from being generated in the first place. Ideally, one would want to start from a coarse surface, and progressively refine it, when and as long as necessary. This approach has been followed in [Grosso & Ertl, 1998].

Our deformable surfaces, as described in Section 5.5.1, are also usable for generating progressive isosurfaces with adaptive precision.

2.3.2 Stream surfaces

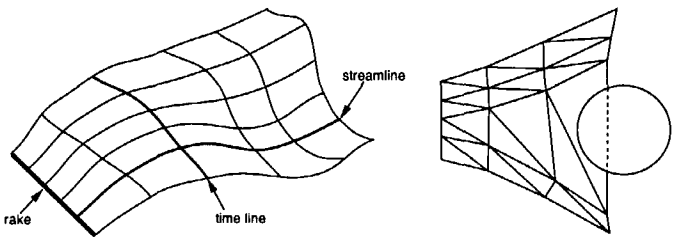
The flow curves in 2D data fields, as defined in Section 2.2, can be extended to *flow surfaces* in 3D data fields. For stationary vector fields in general, the tangent curve is extended to a *tangent surface*, a surface that is everywhere tangent to the vector direction. For stationary velocity fields, a tangent surface is called a *stream surface*. As the velocity direction is everywhere tangent to the stream surface, the velocity component normal to the surface is everywhere zero. This means that no material flows through a stream surface, so it can be considered as a separation between two independent flow zones.

The simplest type of stream surface is a *ribbon*, or a narrow band. Besides local flow direction, it can show the local rotation of the flow. Ribbons can be generated in different ways [Pagendarm & Post, 1997]. First, two adjacent streamlines can be generated from two seed points placed close together, and constructing a mesh of triangles between them. The width of the ribbon depends on the distance between the trajectories of both streamlines, and may become large in a strongly divergent area. A second way is to construct a surface strip of constant width centered around a single streamline. The orientation of the strip is directly linked to the angular velocity of the flow, obtained from the vorticity ω. From the angular velocity, a rotation angle can be found by time integration along the streamline [Pagendarm & Walter, 1994]. The initial orientation is defined at the seed point, and an incremental rotation is applied

in a local coordinate frame at each point on the streamline. The ribbon is constructed by weaving a strip of triangles between the points.

Both methods for creating ribbons have advantages [Pagendarm & Post, 1997]. The first method can show vortical behaviour of the flow, but can also show other effects as well. For instance, it will also show divergence, through the varying width of the ribbon. The second method shows purely local vortical behaviour on the central streamline. In both cases, the surface is not an exact stream surface, as the tangency condition is only true for the constructing streamlines.

A general stream surface can be constructed by generating streamlines from each of a number of points on an initial line segment or *rake*. If for all these streamlines a single constant time step is used, then the lines connecting points of equal time on all streamlines are time lines. Streamlines and time lines thus make a quadrangular mesh (see Figure 2.2a), which can be easily divided into triangles for visualization.



(a) Mesh for stream surface, with streamlines from a rake, and timelines

(b) Divergent flow splits a stream surface

Figure 2.2: Stream surface generation

This standard algorithm has some disadvantages. If the flow is strongly divergent, adjacent streamlines will move too far apart. If there is an object in the flow, the surface must be split, and this is problematic with the standard algorithm (see Figure 2.2b). Finally, if there are high velocity gradients in the flow direction, the mesh will be strongly distorted and unequal-sized and poorly-shaped triangles will result.

To solve these problems, an 'advancing front' algorithm has been proposed [Hultquist, 1992]. The surface is generated in the transverse direction by adding a strip of triangles to the front. Using adaptive time steps to compensate the gradients in the flow direction, all points at the front will move forward by about the same distance. Also, if two adjacent points on the front move too far apart by divergence, a new streamline is started at the midpoint between them. Conversely, if two points move too close together, one streamline is terminated. If an object in the flow is detected, the front can be split, and the two parts can move on separately.

Stream surfaces can also be used to generate streamlines [Kenwright & Mallinson, 1992a]. Two local stream surfaces are determined from dual stream functions defined

in a grid cell. The intersection curve of these stream surfaces is a streamline segment, which is approximated by determining the entry and exit points of the streamline segment in the cell, and connecting these points. A stream line is constructed by tracking from a starting point through the cells. This technique is very different from the time stepping integration techniques described in Section 3.1, and has the advantage of conforming with the physical law of mass conservation.

2.3.3 Implicit surfaces

Two important approaches to mathematically define curves and surfaces are *parametric* and *implicit* [Bloomenthal, 1997]. The parametric approach defines each of the coordinates as an explicit function of one or more parameters. For a 2D surface in 3D space, these functions are: $\mathbf{x} = f_x(s,t), \mathbf{y} = f_y(s,t)$, and $\mathbf{z} = f_z(s,t)$, when (s,t) is an ordered pair of parameters. In the *implicit* approach, the coordinates (x,y,z) are not given explicitly, but as the set of points $(x,y,z) : f(x,y,z) = C$, where C is some constant. Then, $f(x,y,z) = 0$ is a surface implicitly defined by f, or an *implicit surface*, and depending on the form of f, other values of the constant C indicate the distance to the surface. An example is the spherical surface given by $x^2 + y^2 + z^2 = 0$.

Implicit surfaces can be used to generate stream surfaces [van Wijk, 1993]. The stream surface must satisfy the condition $\nabla f \cdot \mathbf{v} = 0$, which means that the normal to the surface (denoted by the gradient ∇f) is perpendicular to the velocity direction. The function f is called the *stream surface function*. The values of f are specified at the inflow boundaries of the flow area, and for all other grid points the values of f are calculated numerically. This can be done in two ways: by solving the convection equation, and by tracing backwards from each grid point to the inflow boundary. A stream surface is then generated as an isosurface of f. This technique can also be adapted to generate time surfaces.

There are two problems with implicit surfaces: it is difficult to specify and control their shape, and it is difficult to sample them uniformly. One solution to these problems, described in [Witkin & Heckbert, 1994], works by placing particles on the surface. Constraints are imposed on the surface and the particles for two purposes: first, it makes the surface follow the particles when they are moved. In this way, the user can control the shape by interactively moving the particles, and the surface will adapt its shape. Second, when the constraint is used in the other direction, it makes the particles follow the surface. In that way, when the surface shape is changed, the particles, which are floating but restricted to the surface, will evenly redistribute themselves over the surface. Note that this is a type of surface consisting of independent points, not connected by edges and polygon faces.

2.4 Deformable models

A special class of geometries, which includes both curves and surfaces, is formed by what we call *deformable models*, geometries which all have in common that they start

from some initial shape, and are deformed in an iterative process to some target shape. This target shape is usually some object in a 2D or 3D image.

The advantage of deformable models is that they allow for the extraction of features which cannot be expressed as isosurfaces, but for which optimization criteria are easy to specify, e.g. curves or surfaces which are located at local gradient maxima. By utilizing a priori knowledge, deformable models may be given properties which isosurfaces do not have. For example, deformable models may be prescribed to have a certain topology, while the topology of a collection of isosurface polygons is usually unpredictable.

Basically, there are three types of deformable models: *deformable curves*, *deformable surfaces*, and *deformable volumes*. Deformable curves are curves in 2D space, deformable surfaces are surfaces in 3D space, and deformable volumes are solid objects in 3D space. The essential difference between deformable surfaces and deformable volumes is that the latter can also represent the internal structure of the domain.

Deformable models have been applied in various areas, such as image processing, medical imaging, and computer graphics. For a good and recent textbook on deformable models, which covers applications in these three areas, see [Metaxas, 1997]. For a recent survey of the use of deformable models in medical imaging, see [McInerney & Terzopoulos, 1996].

We have developed a type of deformable surfaces for a totally different application area: detection of surface features in fluid flows. This will be described in Chapter 5 of this thesis. Since our technique has most in common with deformable curves and surfaces, the remainder of this section will describe related work on deformable curves (see Section 2.4.1) and deformable surfaces (see Section 2.4.2).

2.4.1 Deformable curves

Deformable curves were first proposed in [Kass *et al.*, 1988] for object segmentation in image processing applications. Typically, the user defines an initial curve, in the neighbourhood of the target object to be segmented. The user may put constraints on some points, e.g. by anchoring them to fixed points, or by attaching springs between different points. Then, a minimization process of the curve energy causes the snake to be gradually attracted to the contour of the object. Due to their shape and wiggling behaviour, these curves were called "*snakes*".

In [Kass *et al.*, 1988], snakes were defined as "energy-minimizing splines guided by external constraint forces and influenced by image forces that pull them toward features such as lines and edges". The energy of a snake, which is defined as a parametric curve with parameter s, is defined in [Kass *et al.*, 1988] as:

$$\begin{aligned} E^*_{snake} &= \int_0^1 E_{snake}(\mathbf{v}(s))ds \\ &= \int_0^1 \left(E_{int}(\mathbf{v}(s))ds + E_{image}(\mathbf{v}(s))ds + E_{con}(\mathbf{v}(s))\right) ds \end{aligned} \tag{2.1}$$

where $\mathbf{v}(s) = (x(s), y(s))$ are the positions of the snake elements as a function of the curve parameter s, E_{int} is the internal spline energy, E_{image} is the image energy which causes a snake to be attracted to images features, and E_{con} is the constraint energy imposed by the user.

The internal energy E_{int}, which controls the continuity of a snake, is defined as a weighted average: $E_{int} = (\alpha(s)|\mathbf{v}_s(s)|^2 + \beta(s)|\mathbf{v}_{ss}(s)|^2)/2$. It consists of a first-order derivative term $\mathbf{v}_s(s)$ and a second-order derivative term $\mathbf{v}_{ss}(s)$. The first term makes the snake first-order continuous, the second makes it second-order continuous, with $\alpha(s)$ and $\beta(s)$ determining the weight of each term. If $\beta(s) = 0$, the snake is second-order discontinuous.

The image energy E_{image} causes the snake to be attracted to lines, edges, and (line segment) terminations, and is also defined as a weighted average: $E_{image} = w_{line}E_{line} + w_{edge}E_{edge} + w_{term}E_{term}$.

The last term of the snake energy, the constraint energy E_{con}, allows the user to fix certain points of a snake to the background, or to connect two points of adjacent snakes to each other with a spring. If two points have coordinates $\mathbf{x}_1$ and $\mathbf{x}_2$, then a spring between those two points could add a term $-\kappa\|\mathbf{x}_1 - \mathbf{x}_2\|^2$ to the constraint energy, where κ is the stiffness of the spring.

When all the energy terms have been defined and substituted in the total energy (2.1), the energy is minimized by a variational calculus approach. This basically means that new snake positions are calculated as small variations of the old ones, such that the energy gradually decreases at each iteration, and ultimately is minimized. This will cause an initial snake defined around an object to shrink and lock onto the actual object contour.

The snakes article has been the basis for numerous other types of deformable curves. One further development were the *fast active contours* described in [Williams & Shah, 1990]. There are several differences with the original snakes. One difference is that the calculations are not done in continuous R^2 space, but on a discrete grid of pixels, which makes sense for image processing applications. Another difference with the original snakes is the use of a *greedy algorithm* to compute new positions of snake elements. For each of the neighbour pixels of a snake element, the greedy algorithm calculates the effect of moving that element to that pixel on the new energy of the entire contour. The algorithm then moves each element to the neighbour pixel which leads to the lowest energy. One of the advantages of the greedy algorithm is that it works much faster than variational calculus.

Yet another type of deformable curves is the *Discrete Dynamic Contours* described in [Lobregt & Viergever, 1995]. Like previous models, they are based on minimization of an energy, composed of external energy related to image features, and internal energy related to local curvature. In this case, the energy is not minimized by variational calculus or a greedy algorithm, but by means of forces, which are used in Newtonian motion equations to determine the new positions of the nodes. The force on a node i is given by:

$$\mathbf{f}_i = w_{ex}\mathbf{f}_{ex,\mathbf{r}_i} + w_{in,i}\mathbf{f}_{in} + \mathbf{f}_{damp,i} \tag{2.2}$$

where $\mathbf{f}_{ex,\mathbf{r}_i}$ is the radial component (in the direction $\mathbf{r}_i$) of an external force, with an associated weight, $\mathbf{f}_{in}$ is an internal force, with an associated weight, and $\mathbf{f}_{damp,i}$ is a damping force which increases the stability and ensures termination of the iteration. From these forces, the acceleration for the nodes is easily derived using the relation $a = \mathbf{f}/m$, where is m is a constant mass for all nodes. From the acceleration, the velocity is derived, and using the velocity, the new position $\mathbf{p}_i$ of a node i is determined using a numerical integration:

$$\mathbf{p}_i(t) + \Delta t = \mathbf{p}_i(t)\mathbf{v}_i(t)\Delta t \tag{2.3}$$

where Δt is the time interval between subsequent new positions.

There are several important differences between Discrete Dynamic Contours (DDC) and previous models.

1. The first difference is that in DDCs, minimization of the energy is done for each individual node, not globally for the entire contour, hence the name discrete.
2. The second difference is that displacement of the nodes is done only in the radial direction, or the direction normal to the contour, filtering out any tangential displacement components. This is done for two reasons: tangential displacements do not contribute to deformation of the model, and tangential displacements may cause local accumulation of nodes on the contour, which is not desirable.
3. The third important difference is that DDCs perform resampling of the curve and add new nodes, which can make the final contour much larger than the one originally defined.

In principle, deformable contours are limited to 2D images, or 2D slices of 3D data sets. When 3D representations of objects are required, a frequently applied technique connects contours from several stacked slices by filling the intermediate space by polygonal tilings.

Our deformable surfaces, as described in Chapter 5 of this thesis, have in common with the above methods that they are based on a minimization process. They have in common with the Discrete Dynamic Contours that minimization is done per node rather than for the entire surface. For the node displacement directions, we have compared various mechanisms, including a mechanism similar to the above greedy algorithm, and a mechanism which displaces only in the normal direction.

2.4.2 Deformable surfaces

Deformable surfaces are a logical extension of deformable curves: the 1D curves in 2D space are extended to 2D surfaces defined in 3D space. Whereas deformable curves consist of points $\mathbf{x}(s)$ on a parametric curve, deformable surfaces are either parametric surfaces $\mathbf{x}(u, v)$, or unstructured triangle meshes, which are less straightforward to parameterize.

One example of deformable surfaces consisting of unstructured triangular meshes are the Geometrically Deformed Models (GDMs) described in [Miller, 1990; Miller *et al.*, 1991]. A GDM utilizes a *cost function* somewhat similar to the energy of active

contours. The cost function $C_i(x, y, z)$ for a node i at position (x, y, z) is defined as:

$$C_i(x, y, z) = a_0 D(x, y, z) + a_1 I(x, y, z) + a_2 T_i \tag{2.4}$$

where:

- $D(x, y, z)$ is a deformation potential field that drives nodes towards the boundary.
- $I(x, y, z)$ is an image term, which tries to stop nodes at the boundary.
- T_i is a topology term, which tries to keep neighbouring nodes together.
- a_0, a_1, a_2 are weighting coefficients.

All costs and weights are nonnegative.

The *deformation potential* $D(x, y, z)$ is a global scalar field with values based on a frame of reference, which may e.g. be a point inside the feature to be modelled. The deformation potential must decrease (or increase) monotonically away from a frame of reference, and will repel (or attract) the model away from (or towards) its frame of reference. An example of a localized deformation potential causes each node to be attracted to a point in the direction of the local surface normal. This is comparable to the Discrete Dynamic Contours of the previous section.

The *image term* $I(x, y, z)$ is a scalar field which counterbalances the deformation potential. It identifies transitions from regions in the field which could be a feature from regions which are definitely not a feature. By doing so, it creates a local minimum at boundaries. Operations in image processing that can identify boundaries include digital gradients [Castleman, 1996], and morphological operators [Serra, 1982]. An example of an image term is the shifted threshold operator:

$$I(x, y, z) = \begin{cases} 0 & Image(x, y, z) < T \\ Image(x, y, z) - T & Image(x, y, z) \geq T \end{cases}$$

An image voxel that is part of the object returns the amount it exceeds the object, other voxels return zero. In effect, nodes will approach the boundary of an object, but will be prevented from exceeding it.

The *topology term* T_i maintains the topological integrity of the model. This is necessary because the previous two terms are not enough for a proper growth of the model: if the boundary of an object is incomplete, nodes of an object may 'leak' through the holes, or if the image event detector is noisy, nodes may stop prematurely at an incorrectly detected boundary. Therefore, the topology term tries to keep neighbouring nodes together. An example of a topology term is the local curvature at a node. In [Miller *et al.*, 1991] this is estimated as the ratio between the distance from the node to the centroid of its neighbours, and the maximum distance of the neighbours:

$$T_i = \frac{\|\mathbf{x} - \frac{1}{n}\sum_1^n(\mathbf{x}_n)\|}{\max(\|\mathbf{x}_j - \mathbf{x}_k\|)}$$

where $\mathbf{x}$ is the position of the current node i, n is the number of neighbour nodes, and $\mathbf{x}_j, \mathbf{x}_k$ are the positions of the neighbour nodes. Minimizing this function will try to minimize the curvature by making the neighbour faces of a node coplanar.

There are several important differences between GDMs and snakes (besides the number of dimensions), which are related to the deformation mechanism:

- GDMs perform a local cost function minimization, for each individual node, while snakes perform a global minimization for the entire contour. In this aspect, they resemble the Discrete Dynamic Contours described in the previous section.
- GDMs require an image event field $I(x, y, z)$ to be defined, which indicates which regions of the field may contain features and which regions definitely do not.

Chapter 5 of this thesis describes a type of deformable surfaces which was inspired by the GDMs. Some similarities are: the use of a cost function and the use of an unstructured triangle mesh. But there are also important differences between our deformable surfaces and snakes or GDMs. One difference is that our cost function does not include an internal smoothing energy or a topology-preserving term. Since our deformable surfaces are primarily intended for a different application area than GDMs, namely geometry extraction from fluid flow fields, we have decided not to build an internal smoothing energy into our surfaces. Another difference is that we have applied a new numerical scheme which works much faster than the traditional ones.

2.5 Vortices

While curves and surfaces are useful for obtaining an overview of flow patterns, scientists are often interested in specific flow features, such as vortices. In fact, vortices are among the most important features of fluid flows in many fields of science and engineering [Banks & Singer, 1994]. In aerodynamics, vortices directly affect the flying characteristics of airplanes [Kenwright & Haimes, 1997]. In turbomachinery design, vortices are to be avoided or minimized during design [Roth & Peikert, 1996]. In oceanography, the evolution of vortices in space and time is important for scientists' understanding of ocean circulations [Banks & Singer, 1994]. In fundamental flow research, the evolution and interaction of vortices are studied because vortices play an important role in the development of turbulence. Therefore, detecting and visualizing vortices is an important topic.

Informally, a vortex may be defined as a swirling flow pattern which will often behave as a coherent structure over time [Robinson, 1991]. Unfortunately, there is no formal definition of a vortex, which makes it difficult to detect them. Therefore, vortex detection methods are often based on heuristic criteria.

Methods for detecting vortices fall into three classes:

- *physical methods*, which use point-based physical quantities to define vortices as regions where these quantities have certain value ranges.
- *algorithmic methods*, which also use physical quantities, but utilize more complex algorithms for producing geometries.
- *geometric methods*, which use region-based, geometric properties of streamlines to detect vortices.

Many fluid dynamicists have worked on *physical methods* for vortex detection, based on observed or theoretical indicators of vortices, i.e. values or ranges of physical quantities associated with the occurrence of vortices. The physical quantities include scalar quantities, quantities derived from the velocity field, and quantities derived from the velocity gradient (= rate-of-deformation) tensor (see e.g. [Hunt *et al.*, 1988; Perry & Chong, 1987; Jeong & Hussain, 1995; Müller *et al.*, 1998]). Brief overviews and comparisons of physical vortex detection criteria are given in [Banks & Singer, 1994] and [Roth & Peikert, 1996]. Unfortunately, none of the physical criteria turns out to work in all cases. In Section 4.1, we also show an experimental comparison of several physical methods.

The second category consists of *algorithmic methods*, which are also based on physical quantities, but do not merely select regions where the quantities have certain properties. The quantities are used for performing more complex operations, such as calculating statistical attributes, or tracking the path of a vortex core. The methods described in [Villasenor & Vincent, 1992; Banks & Singer, 1994; Zhu & Moorhead, 1995] are all two-stage methods for extracting vortices. The first stage extracts the vortex core, while the second stage determines the 'boundary' of a vortex by performing an outward search starting from the core. These methods differ in the quantities and in the strategy they use to determine the vortex core and boundary, but the result is always some tubular structure consisting of closed circular curves connected by polygons.

In [Villasenor & Vincent, 1992] a vortex core is built from line segments, whose directions are determined by statistical operations. Starting from some seed point, about 100 random directions are tried. Inside a cylinder with a random direction, the 'field intensity' is calculated. To get an estimate of the intensity, the authors choose: the mean of the lengths of a large number (several hundreds) of field vectors in the cylinder. The cylinder direction with the largest intensity is taken to be the direction of the vortex core segment. All these segments then form a vortex core. A vortex tube is formed by constructing short cylinders of a constant diameter around the core segments. Obviously, this is a fairly coarse description of the boundary.

In [Banks & Singer, 1994], a vortex core is determined as follows. First, points of minimum pressure and maximum vorticity magnitude are chosen as seed points. These seed points are the initial points for a 'predictor-corrector' integration of the vorticity field. At each integration step, the integrated vorticity field predicts a position, which is corrected by moving to the pressure minimum in a plane perpendicular to the vorticity direction. In that way, a series of points is found which form the vortex core. Once the core has been found, closed curves described by splines are used to determine the vortex boundary. This representation is compact yet allows for a wide range of contours.

In [Zhu & Moorhead, 1995], a technique was described to extract ocean eddies, which may be regarded as a specific form of vortices. In contrast to the previous techniques, which were 3D techniques, this is a 2D layered technique, specifically intended for 2D layered data sets common in hydrodynamic applications (see also [Sadarjoen *et al.*, 1998b] and Section 3.3 of this thesis). To find vortex cores, critical points are determined of the velocity field of a 2D layer. To find vortex boundaries, deformable elliptic

contours called Simplified Geometric Deformable Models (SGDMs) are laid around the critical points and expanded. The expansion criterion is determined by maximizing the angle between the contour tangent and the local velocity direction. In this way, a set of elliptic contours for each slice is obtained, which are matched to connect those contours belonging to the same core.

Recently, a third category was developed, the *geometric methods*, which are independent of physical criteria. Typical of these methods is that they only use geometric properties of flow curves to determine whether a region contains a vortex, without employing physical quantities.

In [de Leeuw & Post, 1995], an interactive technique was described for detecting vortices, using a box-shaped region in which sample points were taken. For all the sample points in the box, a number of properties were calculated, including the curvature centre of the streamline through the sample point. If the box contained a vortex, the curvature centres would accumulate near a point, otherwise they would be scattered.

In [Portela, 1997], a rigorous mathematical framework was developed to formalize the intuitively clear concept that a vortex consists of swirling motion around a central set of points. To define swirling motion, the winding-angle concept from differential geometry was used. To define a central set of points, closed Jordan curves were used, which separate the Euclidian plane into the inside and the outside of a vortex. As the Jordan curves and the Euclidian plane are only defined in 2D, this method is limited to detecting vortices in 2D.

A simpler, yet effective method is described in [Sadarjoen *et al.*, 1998b] and in Section 4.5 of this thesis. This method determines the circularity of a streamline or pathline, by measuring the number of circular windings the line makes. If, in addition, the end point of the streamline or pathline is close to its starting point, then the line could be part of a vortex. This method is currently also limited to 2D, but looks promising. Another advantage is that this method also allows for quantification of vortices, by calculating numerical attributes of them.

Related to vortices are *recirculation zones*, which is a frequently-used term indicating regions of separated flow [Shih & Ho, 1994; Chein, 1990]. Separated flows are of great importance, not only from a theoretical point of view, but also in engineering applications, as they occur behind flame holders in combustors, in diffusers, on airfoils, etc. Because recirculation zones often show similar rotational patterns as vortices, it seems to make sense to apply similar detection methods for finding the cores of recirculation zones as for vortex cores.

In this thesis, vortex detection is mainly covered in Chapters 4. Section 4.1 gives a brief overview of experiments with physical methods, but more attention is given to the geometric methods, which are described in Sections 4.3 through 4.5. In addition, Section 5.5.2 of this thesis describes a method for extracting recirculation zones, and Section 6.4.2 for extracting vortices using deformable surfaces.

Chapter 3

Particle Tracing

海浪如雲去卻回
北風吹起數聲雷
朱樓四面鉤疏箔
臥看千山急雨來[1]

Particle tracing is an important technique for visualization of flow fields resulting from computational fluid dynamics (CFD) simulations [Hin & Post, 1993]. This technique visualizes a velocity field by simulating the release of massless particles in the flow, and calculating their trajectories, or motion paths through the field. A source of complications is the use of irregular grids in CFD simulations, such as curvilinear and unstructured grids. See Section 2.1.2 for a discussion on grid types.

This chapter first covers the fundamentals of particle tracing in Section 3.1 and then goes on to describe the specifics for particle tracing in three different types of irregular grids: structured curvilinear grids in Section 3.2, σ-transformed grids in Section 3.3, and unstructured grids in Section 3.4. Section 3.5 shows some example applications. Finally, Section 3.6 gives a summary and conclusions.

3.1 Fundamentals of particle tracing

The calculation of a particle trajectory is based on a stepwise numerical integration of the velocity field, which may be described by an ordinary differential equation [Sadarjoen *et al.*, 1994]. In stationary, or time-independent velocity fields, the equation is:

$$\frac{d\mathbf{x}}{dt} = \mathbf{v}(\mathbf{x}) \tag{3.1}$$

where $\mathbf{x}$ is the position of the particle, t is time, and $\mathbf{v}(\mathbf{x})$ the velocity field. In instationary, or time-dependent velocity fields, the equation is:

$$\frac{d\mathbf{x}}{dt} = \mathbf{v}(\mathbf{x}, t) \tag{3.2}$$

Throughout this thesis, we assume we have stationary flow fields, unless mentioned otherwise. The starting position $\mathbf{x}_0$ of the particle provides the initial condition at

[1] Sea waves come and go like clouds // The north wind starts blowing with thunder // Around the tower, all screens have been raised // lying down I look at the rain on 1000 mountains (Zeng Gong)

initial time t_0: $\mathbf{x}(t_0) = \mathbf{x}_0$. Subsequent points are calculated as

$$\mathbf{x}(t_{i+1}) = \mathbf{x}(t_i) + \int_{t_i}^{t_{i+1}} \mathbf{v}(\mathbf{x})dt \tag{3.3}$$

using a standard numerical integration method, such as Euler, Heun or Runge-Kutta-4.

The solution is a sequence of particle positions $(\mathbf{x}(t_0), \mathbf{x}(t_1), \ldots)$ at time steps $t_0, t_1, \ldots, t_n$. For extensive descriptions of the numerical aspects, see e.g. [Buning, 1989; Hamann *et al.*, 1995; Teitzel *et al.*, 1997].

The above is straightforward when the fields are given in analytical form. However, in scientific visualization, the data is typically given in discrete space, on grids, as described in Section 2.1.2. The structure of a particle tracing algorithm for uniform (regular rectangular) grids may be given by the following pseudo code:

```
find cell containing initial position          (point location)
while particle in grid
    determine velocity at current position     (interpolation)
    calculate new position                     (integration)
    find cell containing new position          (point location)
endwhile
```

Two important components are *point location* and *interpolation*. Point location is the process of determining for a specified point, in which cell it is located, and what its relative position in that cell is. In structured grids, these are denoted by the cell indices (i, j, k) and local offsets (α, β, γ). In uniform rectilinear grids, consisting of cubic cells, these may be easily determined. Let the grid's origin be at (x_0, y_0, z_0) and let the cells be cubes of size s; then, position (x, y, z) is located in the cell with indices:

$$i = \frac{\lfloor x - x_0 \rfloor}{s}, j = \frac{\lfloor y - y_0 \rfloor}{s}, k = \frac{\lfloor z - z_0 \rfloor}{s}$$

and local offsets:

$$\alpha = \frac{\mathrm{frac}(x - x_0)}{s}, \beta = \frac{\mathrm{frac}(y - y_0)}{s}, \gamma = \frac{\mathrm{frac}(z - z_0)}{s}$$

Interpolation is the process of determining a data value at an arbitrary position in a given cell, using the surrounding grid nodes and the local offsets. In uniform grids, this is typically done with first-order trilinear interpolation. Let the velocities at the corner nodes of a cell be denoted by $\mathbf{v}_{000}$, $\mathbf{v}_{001}$, $\mathbf{v}_{010}$, ..., $\mathbf{v}_{111}$. Then, the trilinearly interpolated value is:

$$\begin{aligned} \mathbf{v}(\alpha, \beta, \gamma) \quad = \quad & \mathbf{v}_{000} \cdot (1-\alpha)(1-\beta)(1-\gamma) + \mathbf{v}_{100} \cdot \alpha(1-\beta)(1-\gamma) \\ & + \mathbf{v}_{001} \cdot (1-\alpha)(1-\beta)\gamma + \mathbf{v}_{101} \cdot \alpha(1-\beta)\gamma \\ & + \mathbf{v}_{010} \cdot (1-\alpha)\beta(1-\gamma) + \mathbf{v}_{110} \cdot \alpha\beta(1-\gamma) \\ & + \mathbf{v}_{011} \cdot (1-\alpha)\beta\gamma + \mathbf{v}_{111} \cdot \alpha\beta\gamma \end{aligned} \tag{3.4}$$

3.2 Particle tracing in curvilinear grids

3.2.1 Curvilinear grids

In practice, many CFD applications do not use uniform grids, but *structured curvilinear grids*, consisting of deformed, hexahedral cells, with curved faces (see also Section 2.1). An advantage of curvilinear grids is their ability to conform to the shape of curved or complex geometries, such as airplane wings and coast lines. Another advantage is their regular topological structure, so the cells and data are addressable through indices (i, j, k). A disadvantage of these grids is that algorithms working in them become more complex, because the cells are no longer cubes. Figure 3.1 shows an example of a curvilinear grid. More information about this grid and the corresponding data set are given in Section 3.5.1.

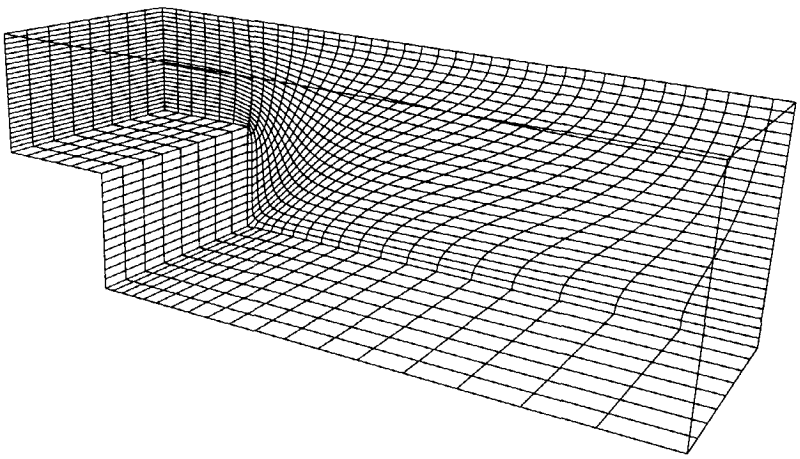


Figure 3.1: Curvilinear grid of a 3D Backward-Facing Step (see Section 3.5.1)

A strategy often applied in many CFD simulation systems works by transforming the curvilinear grid to a uniform rectilinear grid in a new domain. The grid and data, which are typically specified in the 'normal' domain called *physical space*, or P-space, are then transformed to a new domain called *computational space*, or $\mathcal{C}$-space. The integration steps are done in $\mathcal{C}$-space, and the resulting positions are transformed back to $\mathcal{P}$-space. Unfortunately, for particle tracing algorithms, this method often leads to

a decrease in accuracy and efficiency, as was investigated in detail in [Sadarjoen *et al.*, 1994; Sadarjoen *et al.*, 1997].

In brief, it turned out that the transformation of the vector field, when done in a simple way, introduced large errors, but when done in an accurate way, caused an enormous increase of storage. For an accurate transformation, each vector defined at a grid node in $\mathcal{P}$-space has to be transformed to eight different vectors in $\mathcal{C}$-space, depending on which cell the node is considered to be part of. It also turned out that the transformations from $\mathcal{P}$-space to $\mathcal{C}$-space, and back to $\mathcal{P}$-space again, were more costly than point location directly in $\mathcal{P}$-space.

Another strategy works by calculating the particle path directly in the curvilinear grid in the normal domain, $\mathcal{P}$-space. This avoids transformations between the two domains, although at the expense of more difficult point location. This is the case because there is no longer a direct relation between the coordinates of a point and the cell indices. Instead, a search must be performed in several cells, to check which of them contains the point. In addition, in curvilinear grids it is harder to determine the local offsets, i.e. the local position inside a cell.

3.2.2 Tetrahedral 5-decomposition

One way to cope with curved cells works by decomposing the hexahedral cells into tetrahedra. The advantages of tetrahedra is that they are convex and planar, which facilitates containment tests and face intersection tests.

The simplest and most efficient scheme is to decompose the hexahedral cells into five tetrahedra, henceforth called the *5-decomposition* [Sadarjoen *et al.*, 1994; van Walsum, 1995]. This method was later adopted in [Kenwright & Lane, 1995]. Figure 3.2a shows a cube which is decomposed into one central tetrahedron and four corner tetrahedra. In a structured grid, the decomposition can be done in two orientations. To guarantee connection of cell faces and to avoid overlapping cells, these two orientations should be alternated in adjacent cells, as shown in Figure 3.2b.

In tetrahedra, *interpolation* is typically performed using linear interpolation. Figure 3.3 shows a tetrahedron ABCD, where α, β, γ denote the local offsets in the tetrahedron, with the restriction that $\alpha + \beta + \gamma \leq 1$. If $\mathbf{v}_A$ is the data value in node A, $\mathbf{v}_B$ the data value in node B, etc., then the interpolated value $\mathbf{v}_P$ in some position P in the tetrahedron is:

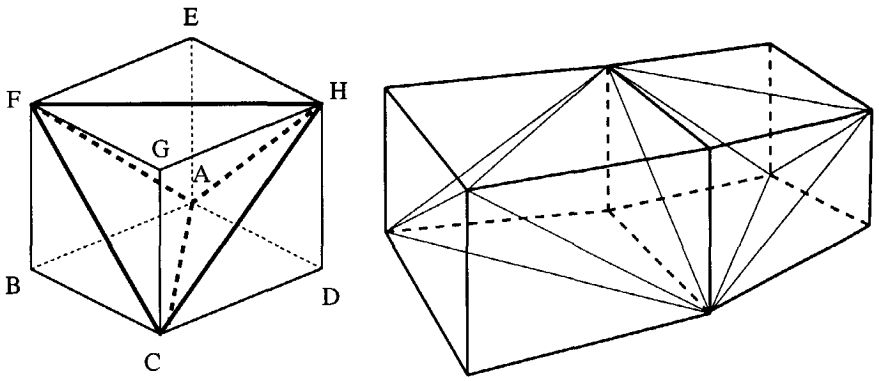
$$\mathbf{v}_P = \mathbf{v}_D + \alpha(\mathbf{v}_A - \mathbf{v}_D) + \beta(\mathbf{v}_B - \mathbf{v}_D) + \gamma(\mathbf{v}_C - \mathbf{v}_D)$$

The local offsets (α, β, γ) may be found by inverting the interpolation of the known position of P in the tetrahedron:

$$P = D + \alpha(A - D) + \beta(B - D)\gamma(C - D) \tag{3.5}$$

$$(\alpha, \beta, \gamma) = (A - D|B - D|C - D)^{-1}(P - D) \tag{3.6}$$

Point location is done as follows: a line is drawn between the previous, known position and the new, unknown position. Along this line, intersections are calculated



(a) one central tetrahedron and four corner tetrahedra

(b) Alternating orientations for tetrahedral decomposition

Figure 3.2: Tetrahedral 5-decomposition of a hexahedral cell

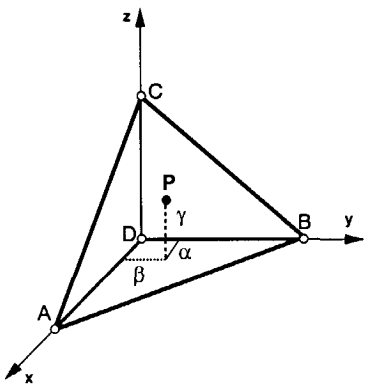


Figure 3.3: Linear interpolation in a tetrahedron

between the line and the faces of the tetrahedra, into which the hexahedral cells are decomposed. By determining which faces are intersected, it can be determined which adjacent tetrahedra are traversed. For each traversed tetrahedron, a point-in-polyhedron test is performed, until the line endpoint is in the current tetrahedron, at which point the unknown cell has been found and the point location is complete. The decomposition is done on the fly, only for cells traversed, thus avoiding the overhead of a global tetrahedrization.

3.3 Particle tracing in σ-transformed grids

3.3.1 σ-transformed grids

Point location using tetrahedral 5-decomposition regularly fails in a specific type of grids known as *σ-transformed grids* [Sadarjoen, 1994; Sadarjoen *et al.*, 1998a]. In our test cases, up to 40%(!) of the particles were caught in an infinite loop between two cells, or stopped completely. Before explaining the cause of these problems, let us first describe this type of grids.

σ-transformed grids are commonly used in hydrodynamic simulations of shallow waters, such as marine coasts or estuaries. They consist of stacked 2D xy-layers, each of which is a well-formed quadrangular mesh with curved and approximately orthogonal grid lines. Figure 3.4 shows an example. Corresponding nodes in different layers have identical x,y-coordinates.

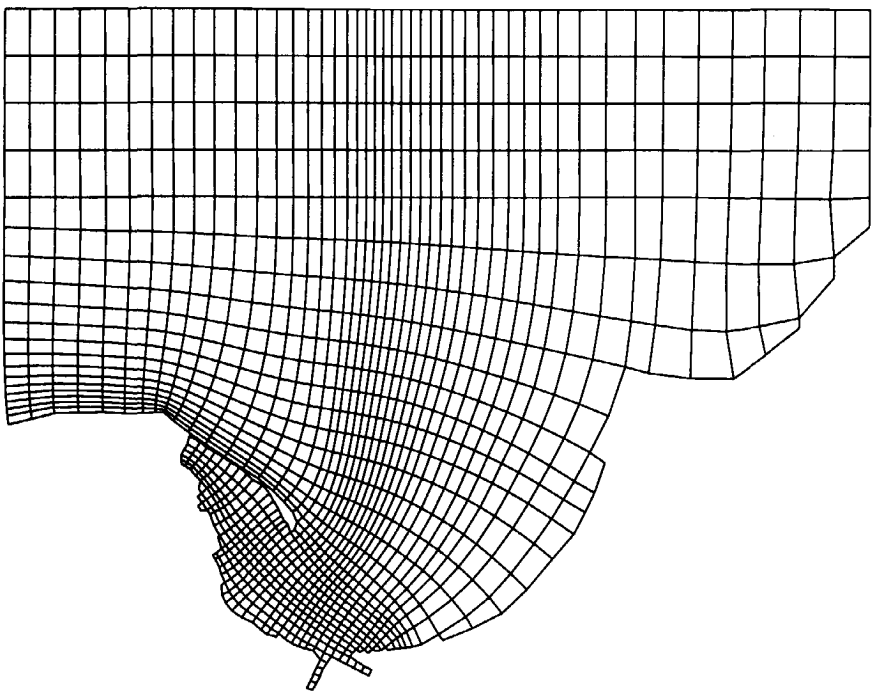


Figure 3.4: Horizontal slice of curvilinear σ-transformed grid of the Bay of Gdańsk (see Section 6.2). Data courtesy WL | Delft Hydraulics

In the vertical direction, the grid lines are straight and parallel to the z-axis. σ-

coordinates are defined relative to the local water elevation ζ and depth d, as $\sigma = \frac{z-\zeta}{\zeta+d}$. The top layer, where $\sigma = 0$, follows the free water surface, which usually only varies gradually. The bottom layer, where $\sigma = -1$, follows the sea bed geometry, which typically has strongly varying depths throughout the model. The layers in between have a prescribed thickness distribution according to the σ-transformation. Figure 3.5 shows one possible distribution with six layers. Figure 3.6 shows a real-life example, with a sea bed geometry and a vertical grid slice of the Lith harbour data set, which was produced at WL | Delft Hydraulics in a simulation as part of the design of a harbour near Lith [Meijer, 1995].

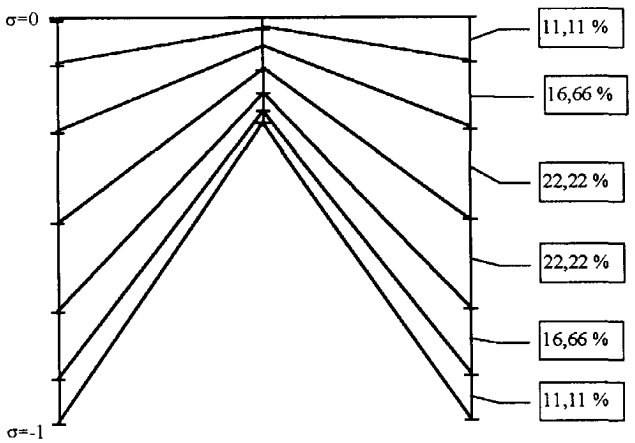


Figure 3.5: Vertical thickness distribution of a six-layer σ-transformed grid

In σ-transformed grids, a considerable number of cells may be sheared in the vertical direction, because the number of layers is constant while the local depth varies, so parallel vertical edges often lie at very different depths. The shearing is increased by the cells typically being very thin in these applications: the model may be hundreds of kilometers wide yet only tens of meters deep.

If the amount of shearing is high and the cells are very thin, the 5-decomposition may become degenerate. Whereas in normal cells the orientation of the central tetrahedron is as shown in Figure 3.7a, in strongly sheared cells the orientation of the central tetrahedron is reversed ('turned inside out'), as seen in Figure 3.7b. The top faces BEG and DEG now lie at the bottom, while the bottom faces BDE and BDG lie on top. The edges BD and GE have crossed each other. This is possible because the central tetrahedron has edges spanning the entire cell.

Let us investigate the process of reversal between the normal and the degenerate state of the central tetrahedron in more detail. Figure 3.8 shows the same cell as Fig-

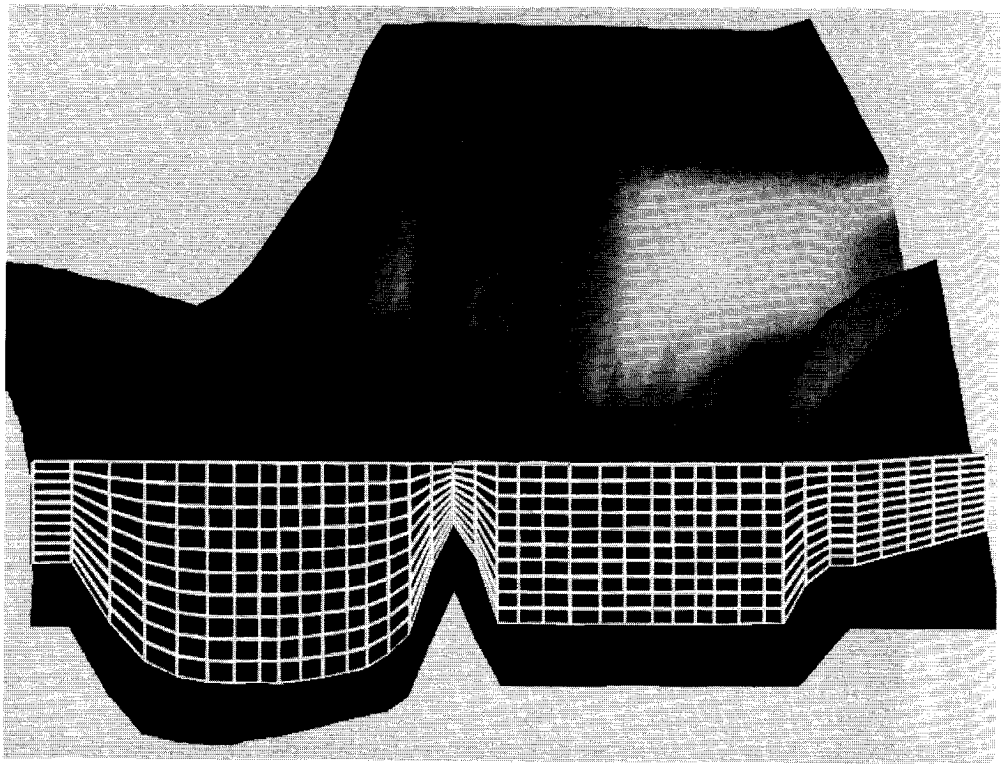
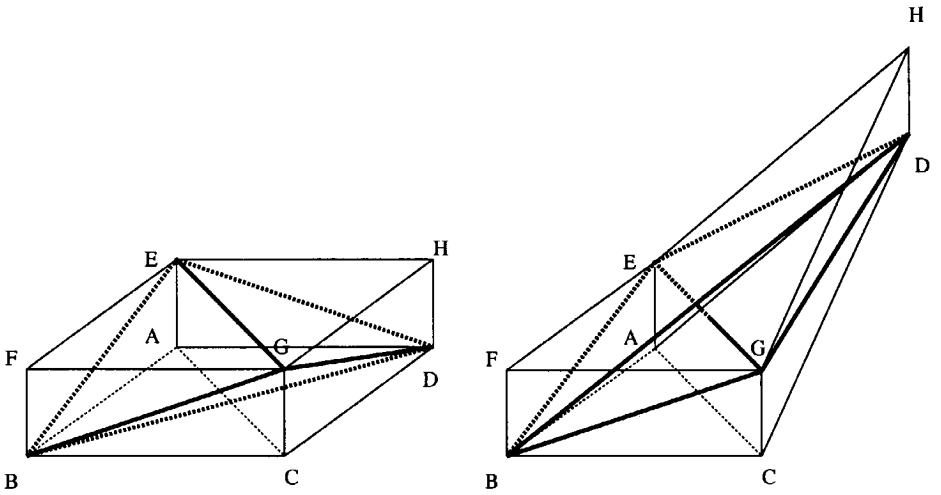


Figure 3.6: Ten-layer σ-transformed grid of Lith Harbour, with a river bed geometry and a vertical grid slice. Data courtesy WL | Delft Hydraulics

ure 3.7, but now the four faces of the center tetrahedron each have a different colour (grey shade). The subfigures show different degrees of deformation, expressed in the height h over which edge DH is displaced; $h = 0$ means no deformation. In Figure 3.8a, $h = 0$ and the upper (black and white) faces are on top. In Figure 3.8b, $h = 1$ and all four faces are coincident, as the black and white faces are visible but a crossing edge of the other two faces is also visible. In Figure 3.8c, where $h = 1.5$, the two lower faces (grey) are on top. The height h for which the central tetrahedron becomes degenerate, depends on the decomposition orientation and on the height of the other vertical edges.

The degenerate 5-decomposition causes two problems. One problem is that particles get caught in an infinite loop between two tetrahedra. Due to the reversed orientation of the central tetrahedron, the point location algorithm finds a false exit face, and therefore a false adjacent tetrahedron. As a consequence, the algorithm moves from a corner tetrahedron to the central tetrahedron, and then returns to the corner tetrahedron where it came from, instead of proceeding to the next one. In this way, it



(a) Normal cell

(b) Strongly sheared cell

Figure 3.7: (a) normal and (b) reversed orientation of the central tetrahedron BDEG

will continue moving back and forth forever.

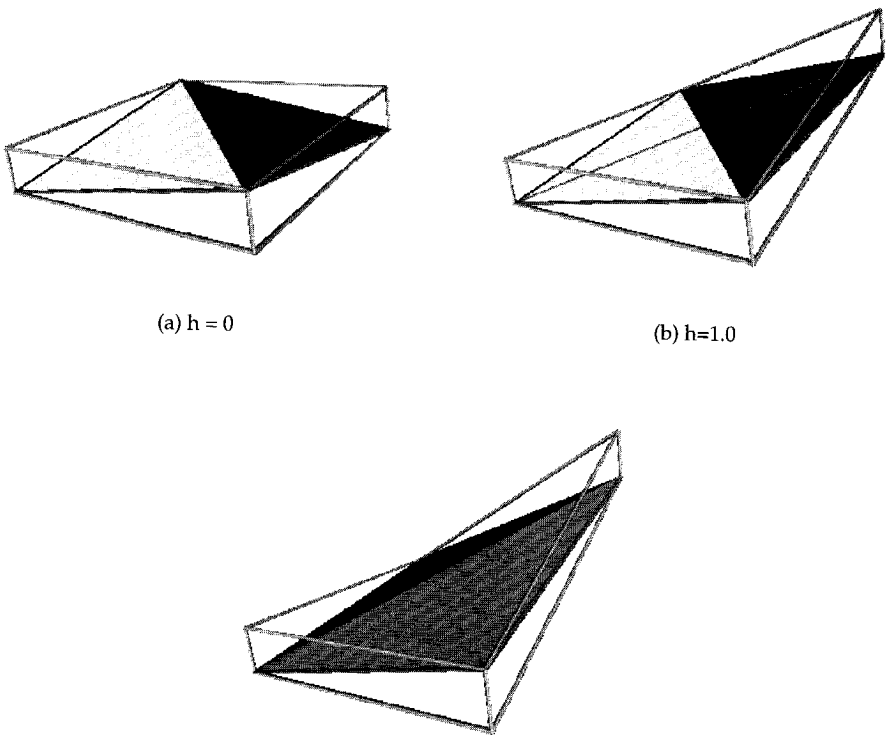
Another problem is that the central tetrahedron may overlap with corner tetrahedra, and even with neighbouring cells. As a consequence, a point location algorithm cannot determine a unique tetrahedron which contains a given point, and fails.

In typical σ-transformed grids, the frequency at which these problems occurred, varied between 4% and 40% of the particles, depending on the data set and particle source locations. In some cells, the problems might be solved by changing the decomposition orientation, since the problem is orientation-dependent. But then the problem would occur in other cells, because the orientation must be chosen globally for the entire grid.

3.3.2 Tetrahedral 6-decomposition

An apparent solution to the point location problems would be to scale or shear a deformed cell such that the reversed orientation of the edges and tetrahedra is avoided. However, scaling or shearing grid cells amounts to applying a computational space algorithm: the grid is transformed to a different domain, where the cells are regular and rectangular. We chose not to do this because of the loss of accuracy and efficiency [Sadarjoen *et al.*, 1997].

A better approach is to use a different tetrahedral decomposition. A systematic overview of all the possibilities, given in [Albertelli & Crawfis, 1997], shows that a



(a) h = 0

(b) h=1.0

(c) h=1.5

Figure 3.8: Cell with increasing degrees of deformation h

hexahedron can be decomposed into 5, 6, or any even number of tetrahedra between 12 and 24. For reasons of efficiency and storage space, the preferred approach is the decomposition into 6 tetrahedra, henceforth called the *6-decomposition*.

Figure 3.9 shows the 6-decomposition: a hexahedral cell is decomposed into two three-sided prisms, each of which is decomposed into three tetrahedra. Just like the 5-decomposition, the 6-decomposition can have two orientations: each face diagonal can be chosen in two ways, but unlike the 5-decomposition, the 6-decomposition method does not require the orientations to alternate for adjacent cells.

The main advantage of this 6-decomposition method is that it solves the point location problems. There is no longer a central tetrahedron with edges which span the entire cell and which cross each other when the cell is sheared in the vertical direction.

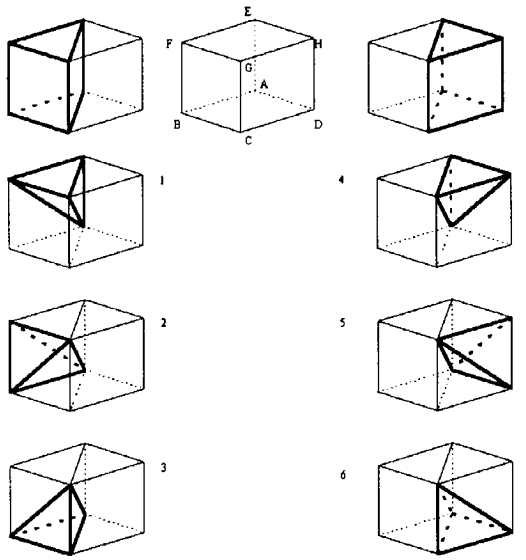
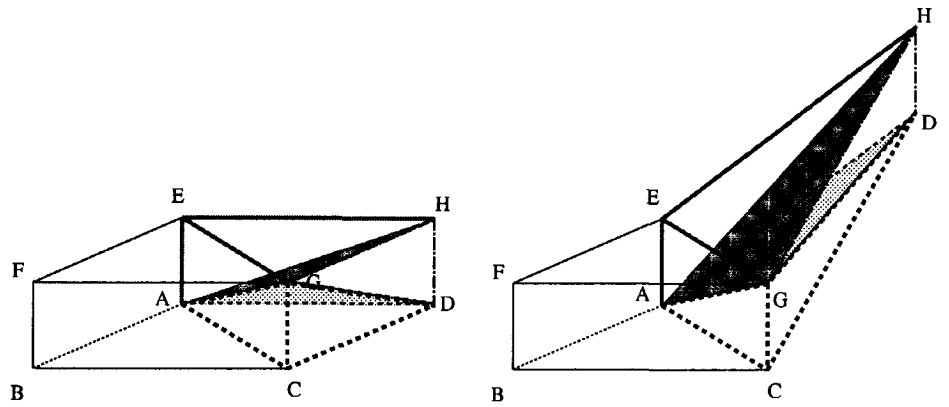


Figure 3.9: Decomposition of a hexahedron into six tetrahedra (compare Figure 3.2)

Figure 3.10 shows one orientation of a 6-decomposition of a normal cell, and in a sheared cell comparable to Figure 3.7. One of the prisms (ACDEGH) has been decomposed into three tetrahedra: ACDG, ADGH, and AEGH. It can clearly be seen that the tetrahedra ACDG, ADGH, and AEGH retain their orientations when the prism is sheared, as the hatched planes AGH and ADG retain their relative positions: none of the three tetrahedra is turned inside out. It can also be shown that the tetrahedra will never change their orientations, no matter how large the shearing is, as long as the edges are only displaced in the vertical direction (as is the case with σ-transformed grids).

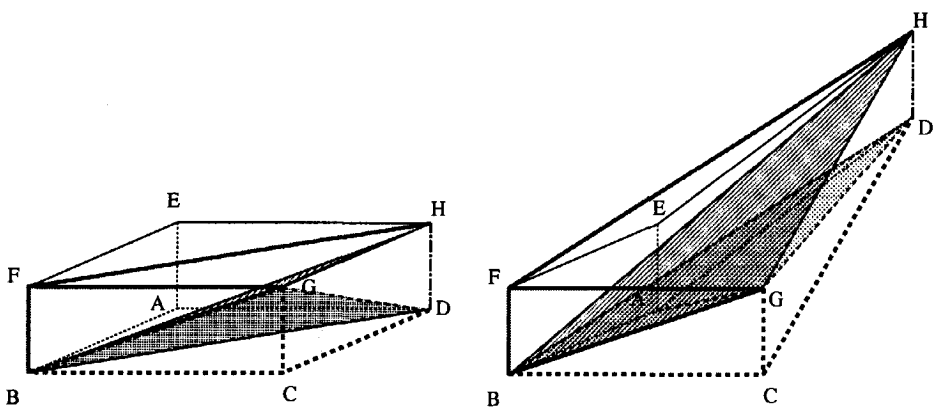
Figure 3.11 shows the other orientation of the 6-decomposition of the same cell. Just like in the previous figure, one of the two prisms (BCDFGH) has been decomposed into three tetrahedra. The prism in front of the plane BDFH is decomposed into the tetrahedron BFGH, which lies on top of the hatched plane BGH, the tetrahedron BCDG, which lies under the hatched plane BDG, and the tetrahedron BDGH, which lies in between. Here, it can also be seen that the tetrahedra retain their orientations when the prism is sheared, since the hatched planes BGH and BDG retain their relative positions.

The performance of the 6-decomposition is similar to that of the 5-decomposition: in tests, it was found that the execution speed of both algorithms was practically the same. In theory, one might expect the 6-decomposition to be slower, because a cell is decomposed into more tetrahedra. However, in practice it was found that a particle



(a) Normal cell (b) Sheared cell

Figure 3.10: One orientation of the tetrahedral 6-decomposition; the tetrahedra retain their relative positions.



(a) Normal cell (b) Sheared cell

Figure 3.11: Other orientation of the tetrahedral 6-decomposition; the tetrahedra retain their relative positions.

typically traverses only 3 of the tetrahedra, regardless of which decomposition is being used.

3.4 Particle tracing in unstructured grids

So far, the grids we have used have been structured grids, which are characterized by a regular grid topology: in 3D space, the grid always consists of $l \times m \times n$ cells, and for each cell it is known which are the neighbouring cells. In contrast, unstructured grids have an irregular topology, so it is not known in advance which neighbours a cell has (see also Section 2.1). While structured grids typically consist of quadrangles in 2D and hexahedra in 3D, unstructured grids often consist of triangles in 2D and tetrahedra in 3D, although unstructured grids of hexahedra also exist. Figure 3.12 shows an example of an unstructured grid, used in a Finite Element Analysis of a slice of a human skull [van den Broek *et al.*, 1998].

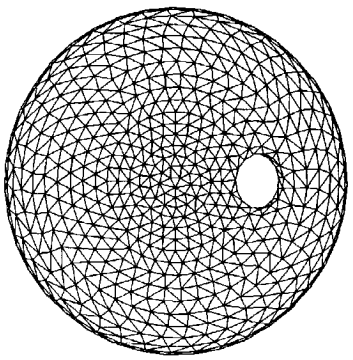


Figure 3.12: Unstructured grid from Finite Element Analysis

An advantage of unstructured grids is the greater degree of modelling freedom. A disadvantage of unstructured grids is the greater complexity of certain algorithmic operations. Especially point location and grid traversal are difficult, because they rely on a stepwise traversal between neighbouring cells, but information about cell neighbours is not available in these grids.

What is therefore necessary, is the addition of extra adjacency information for each cell. One way to do this is to derive and store information about cell faces, such that an algorithm can query the data structure to find the answers to the following questions:

1. which faces belong to a cell?
2. which cells are on either side of a face?

Figure 3.13 shows the standard and additional information that can be added to handle unstructured grids. The standard information typically consists of a list of

nodes with their coordinates, and a list of cells with pointers to the nodes they consist of (cell_nodes). These relations are depicted in the figure by solid arrows.

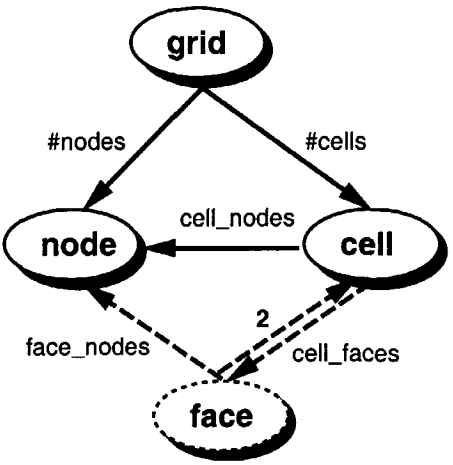


Figure 3.13: Connectivity relations between cells, faces, and nodes in a grid; dashed lines indicate additional connectivity information.

The additional information consists of a list of faces, with pointers to the nodes at their corners (face_nodes), and to the two cells sharing that face. Before the latter can be determined, pointers should first be determined from each cell to its faces (cell_faces). These three relations are depicted in the figure by dashed arrows.

The number of nodes (#nodes) and number of cells (#cells) depend on the size of the data set. The number of nodes per cell (cell_nodes), number of faces per cell (cell_faces), and number of nodes per face (face_nodes) depend on the type of cells the grid consists of. Some possibilities for this are listed in Table 3.1.

cell type	cell_faces	face_nodes	cell_nodes
tetrahedra	4	3	4
hexahedra	6	4	8
4-sided pyramids	5	3 or 4	5
3-sided prisms	5	3 or 4	6

Table 3.1: Connectivity relations in various grid types

A library called `CNX-lib` was implemented with functions which derive and use the additional connectivity information needed to handle unstructured grids [van der Wouden, 1997]. The library supports multiple development environments: AVS-5™, AVS/Express™, and stand-alone C and C++ applications, and provides a uniform way of data access regardless of the development environment. Currently, functions are available for:

- point location
- ray traversal
- interpolation
- gradient calculation

These functions support grids consisting of either tetrahedra or hexahedra, which suffices for most of the unstructured grids currently in use. When needed, more functions could be added to support other cell types as well.

Figure 3.14 shows the layered architecture of `CNX-lib`, and the data flows from the development environment and to the application.

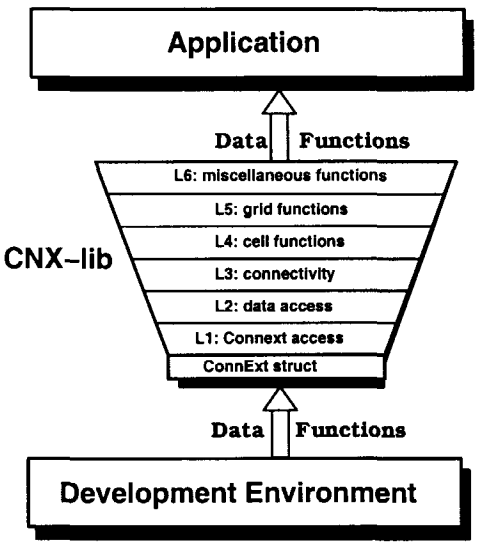


Figure 3.14: `CNX-lib` architecture

The `CNX-lib` consists of a `Connext` (Connectivity extension) data structure and a set of C/C++ functions. The `Connext` data structure contains the derived connectivity data and pointers to the available grid. Deriving this data structure is done in a preprocessing step, as it is time-consuming, but it only needs to be done once for each data set, after which the information can be reused in order to save time.

The C/C++ functions have been grouped into levels, in accordance with the layered architecture:

- Level 1 contains functions to directly access the `Connext` data structure.
- Level 2 contains functions to access the volume data.
- Level 3 contains functions to derive and query the `Connext` data structure.
- Level 4 contains functions for operations at the cell level, such as interpolation.
- Level 5 contains functions for operations at the grid level, such as point location and ray traversal.
- Level 6 contains miscellaneous functions, such as for gradient calculation.

3.5 Examples

The techniques described in the previous sections were implemented in three systems:

- For particle tracing in curvilinear grids, a stand-alone system called PLANKTON was developed [Hin, 1994; Sadarjoen *et al.*, 1994] in C and Graphics Library (GL) by Silicon Graphics. The system handles rectilinear and structured curvilinear grids of hexahedral cells by applying tetrahedral 5-decomposition.
- For particle tracing in σ-transformed grids, a set of AVS/Express modules called PLANKTON-97 was developed [de Boer, 1998]. The system handles σ-transformed grids of hexahedral cells by applying tetrahedral 6-decomposition [Sadarjoen *et al.*, 1998a]. As PLANKTON-97 uses AVS/Express components for the user interface and graphics rendering parts, it is multi-platform: it should work on all platforms capable of running AVS/Express.
- For particle tracing in unstructured grids, an AVS-5 module was developed based upon the CNX-lib library [van der Wouden, 1997]. The system works with unstructured grids of either tetrahedral or hexahedral cells.

The following sections show applications of the above systems for each type of grid.

3.5.1 A 3D backward-facing step

This section illustrates particle tracing in structured curvilinear grids, using a 3D Backward Facing Step (BFS). This is a model of a channel with a step mounted in it, which results in a suddenly increased channel width behind the step. Depending on the flow conditions (e.g. the Reynolds number) and on the step width compared to the channel width, this may result in flow separation just behind the step, and flow reattachment further down the channel. The separated flow which occurs just behind the step is also known as a *recirculation zone*, because the flow exhibits a recirculating pattern there. Figure 3.15 shows a scheme of the flow structure in a 2D BFS. The flow structure in 3D is more complex, because in addition to flow 'within a 2D slice', there may be flow components perpendicular to the slices, so-called 3D components.

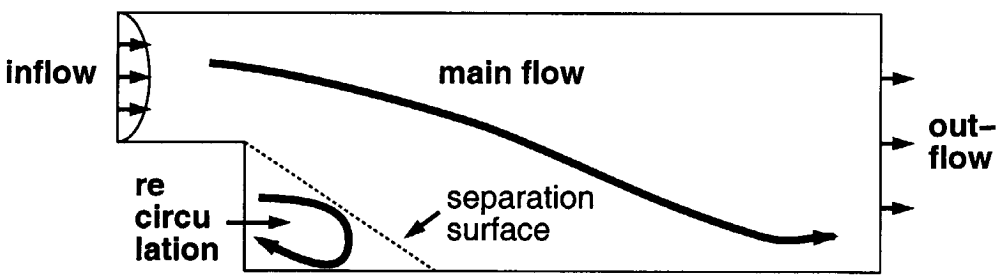


Figure 3.15: 2D Backward-facing step

Many researchers have studied this case, e.g. Chein, who numerically modelled a laminar flow over a 2D BFS, to make predictions about streamline patterns, veloci-

ties, vorticities, and separation and reattachment lines [Chein, 1990]. In [Shih & Ho, 1994], a 3D BFS was studied experimentally, and it was found that the reattachment positions and the flow characteristics inside the recirculation zone were highly three-dimensional, depending on the aspect ratio between channel width and step height.

The data set we use here is a 3D BFS calculated in a numerical simulation performed at the Mathematics Dept. of Delft University of Technology, as one of the test cases for the development of the ISNaS CFD flow solver [Mynett *et al.*, 1991]. Figure 3.1 on page 23 shows the 3D curvilinear grid of this data set, which consists of $25 \times 37 \times 9$ nodes; at each node, pressure and three velocity components were calculated.

The PLANKTON system [Hin, 1994] was used to trace up to 200 time steps of $\Delta t = 0.1s$ of ten particles released along a horizontal line in the inflow. The particle tracing algorithm applied here was a physical space algorithm employing tetrahedral 5-decomposition as described in Section 3.2. This worked without problems, as the grid of this model was a curvilinear grid without strongly sheared cells.

Next, a custom-made AVS-5 module called `render_particles` was used to render the particle paths as tubes and animate them. Figure 3.16 shows two frames from this animation. The paths are coloured with the velocity magnitude. One path, highlighted in white, makes a loop through the recirculation zone. This path also shows the 3D nature of the flow, as its movement is not contained within a 2D slice, but also moves sideways.

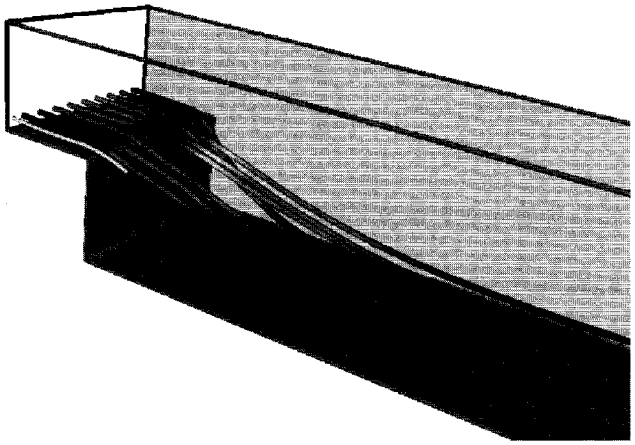
This example shows that the 5-decomposition is an effective method for particle tracing in structured curvilinear grids.

3.5.2 Lith Harbour

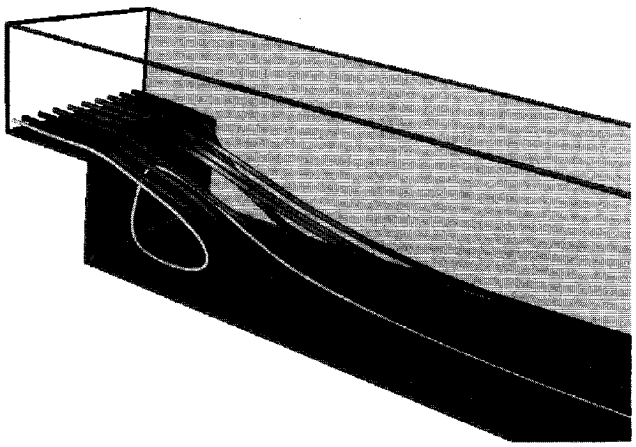
This section illustrates particle tracing in σ-transformed grids. The case we use is a project carried out at WL | Delft Hydraulics which involved the design of a harbour near Lith by the Maas river [Meijer, 1995]. The harbour is separated from the river by a thin dam. The geometry of the harbour was to be designed in such a way that the overall flow in the river was guided smoothly past the entrance, in order to minimize siltation and associated dredging of the harbour entrance due to deposition of river sediment. In evaluating the design alternatives, particle tracing was one of the techniques used [Mynett *et al.*, 1995].

The data set used from this case is a σ-transformed curvilinear grid with $121 \times 40 \times 10$ cells. At the grid nodes, velocity and turbulence intensity are defined. Figure 3.6 on page 28 showed a 3D view of the grid highlighting the σ-transformation. Figure 3.17 shows a 2D horizontal (xy) grid slice consisting of 80×39 nodes. The figure also shows another typical feature of many hydrodynamic grids: land points are not included in the computational model, which results in a topologically incomplete grid. The units along the axes are Parisian longitude and latitude coordinates, expressed in meters from Paris; the extent of the area is approximately 600×600 m.

As PLANKTON requires that the grid be a full array of $l \times m \times n$ points, additional dummy points had to be specified, which results in the grid slice shown in Figure 3.18.



(a) frame 25



(b) frame 200

Figure 3.16: Two frames of a particle path animation in a 3D backward-facing step. The white particle path, which goes through the recirculation zone, shows the 3D nature of the flow.

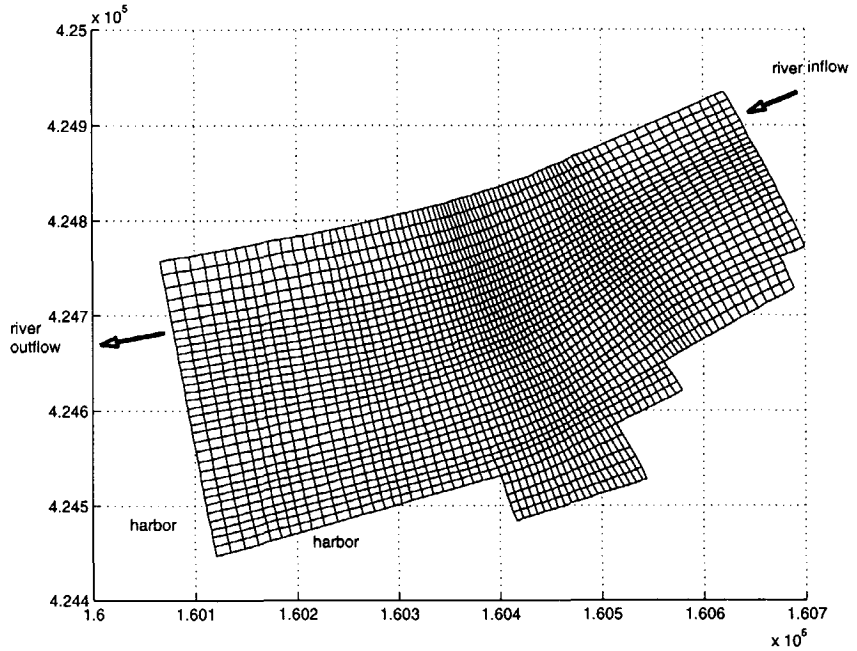


Figure 3.17: Lith Harbour 2D horizontal grid slice. Data courtesy WL | Delft Hydraulics

First, PLANKTON was used to release particles in the upstream part of the river, which is in the upper right in Figure 3.17 and in the harbour, which is in the lower left of the figure. Trajectories were calculated using up to 201 time steps of $\Delta t = 50s$.

An OpenInventor™ application, developed for [Reinders *et al.*, 1999], was used to render the particles as shaded spheres, as well as the river floor geometry. Figures 3.19 and 3.20 show four frames of an animation consisting of 201 frames. Figure 3.19a shows the initial positions of the particles, and the bottom geometry. Here, the view is approximately from the north west, with the harbour being in the upper half of the figure, and the Maas river in the lower half. The river and the harbour are separated by a thin dam. Figure 3.19b shows animation frame 15, where it can be seen that the particles in the river have a much higher velocity than those in the harbour. Figure 3.20a shows frame 22, where some particles in the river have already left the domain. Finally, in Figure 3.20b, only particles in the harbour are left.

The `render_particles` AVS module was again used to render the particle paths as tubes, resulting in Figure 3.21. The cover of this thesis shows a colour version of this figure.

The particle paths in Figure 3.21 clearly show the flow patterns in the river and in the harbour. From the shape of the particle paths, it can be seen that the harbour contains a recirculation zone. From the length and the colours of the paths, it can be

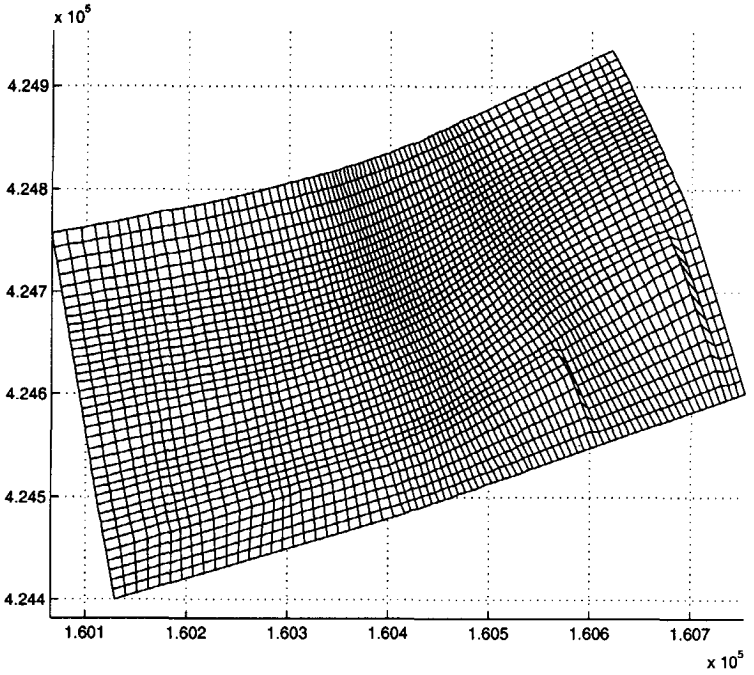
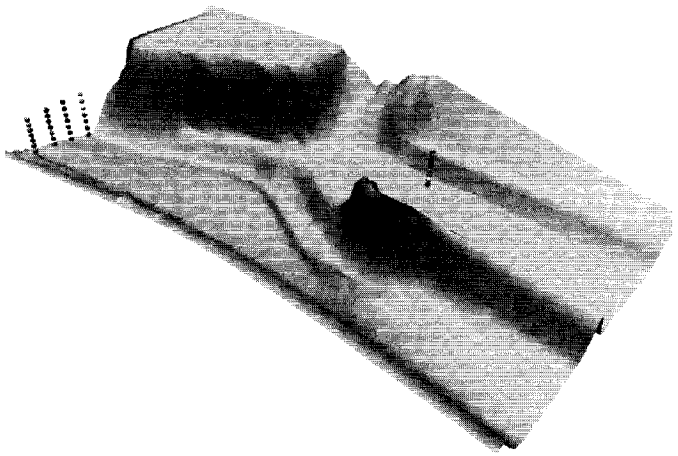


Figure 3.18: Lith Harbour; 2D horizontal grid slice with additional dummy points

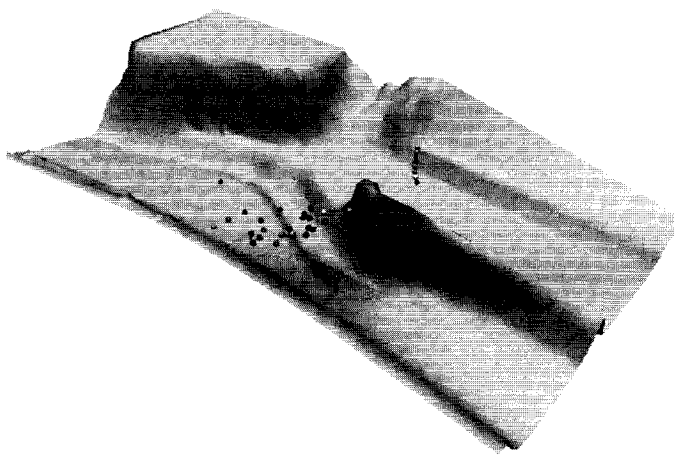
seen that the velocities in the recirculation zone are much lower than in the river.

In another experiment, particles were only released in the upstream part of the river. In this case, PLANKTON, which relies on the 5-decomposition, experienced problems with 14% of the particles, caused by strongly sheared cells [de Boer, 1998], [Sadarjoen *et al.*, 1998a]. In contrast, PLANKTON-97 experienced no problems, as the 6-decomposition proved to be insensitive to strongly sheared cells.

Figure 3.22 shows two frames of a PLANKTON-97 animation, where the particles have been rendered as arrows, coloured with the velocity magnitude. The particles clearly show the velocity profile, as the particles near the river bed are slower than the particles near the water surface.

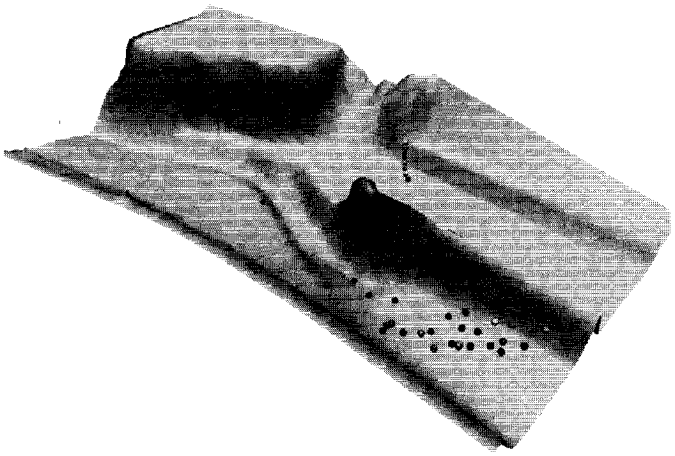


(a) Frame 0

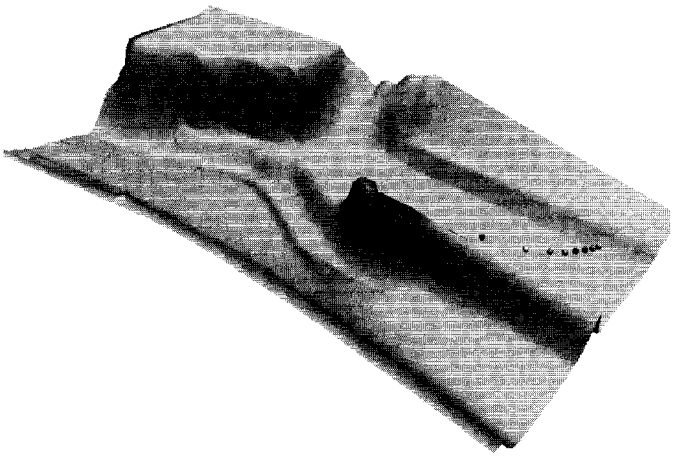


(b) Frame 15

Figure 3.19: Lith Harbour; frames 0 and 15 of a `PLANKTON` animation



(a) Frame 22



(b) Frame 201

Figure 3.20: Lith Harbour; frames 22 and 201 of a PLANKTON animation

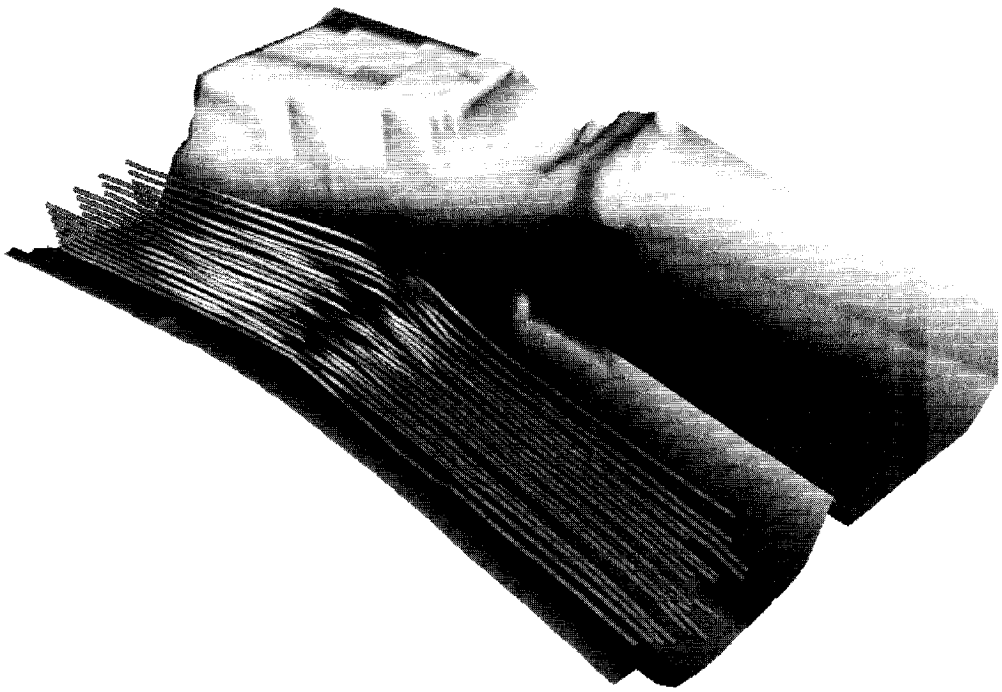


Figure 3.21: Lith Harbour; particle paths calculated with PLANKTON, rendered with AVS-5

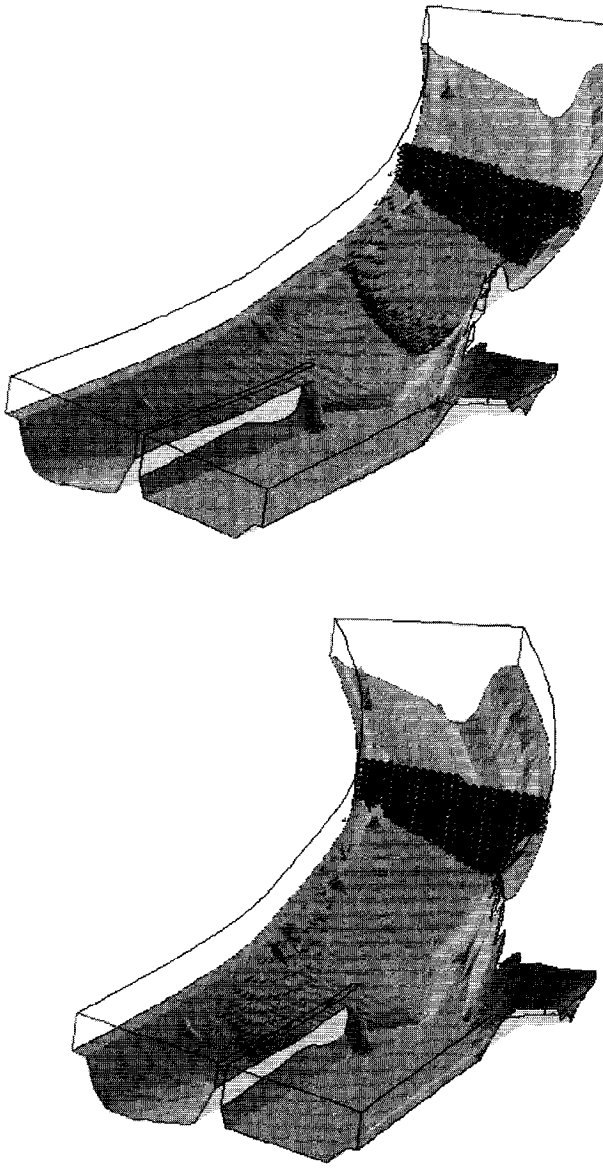


Figure 3.22: Lith Harbour; two frames of a PLANKTON-97 animation

3.5.3 A blunt fin

This section illustrates particle tracing in unstructured grids. The case we use is an unstructured Blunt Fin data set as provided with AVS-5 [AVS, 1993]. This data sets models the flow near the air-intake of a hypersonic aircraft. The data set is defined on an unstructured grid of 640 nodes and 441 hexahedral cells, with each node containing the following quantities: density, x-, y-, and z-momentum, and stagnation. Figure 3.23 shows the grid.

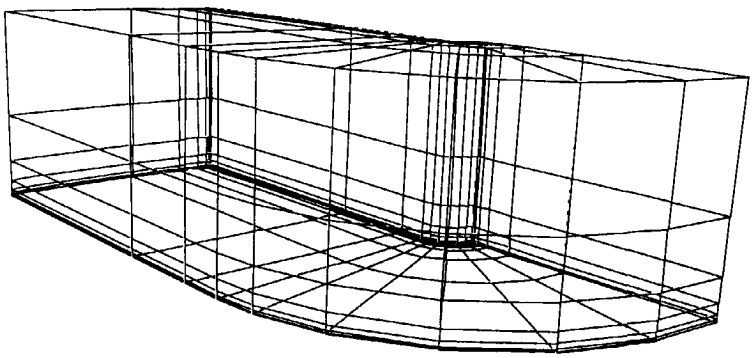


Figure 3.23: Blunt Fin; unstructured hexahedral grid

An AVS-5 module called `avs_cnx_sline` was developed which uses the `CNX-lib` library for data on unstructured grids [van der Wouden, 1997]. With this module, we released particles from a vertical rake and calculated their paths. Figure 3.24 shows these particle paths. A comparison with particle paths calculated by the AVS-5 built-in module shows some slight differences. These are caused by different interpolation methods. AVS-5 uses the value of the nearest-neighbour, while CNX-lib uses linear interpolation of corner node values.

This example demonstrates that the development of the `CNX-lib` library for unstructured grids was successful, which enabled particle tracing in unstructured grids, but the library may also be used for the development of other visualization techniques on unstructured grids.

3.6 Conclusions

In this chapter, we have described techniques for particle tracing in various kinds of grids: structured curvilinear grids, σ-transformed grids, and unstructured grids. Each of these grids poses different requirements for a particle tracing algorithm.

For particle tracing in structured curvilinear grids, which consist of hexahedra, the cells can be decomposed into five tetrahedra. For particle tracing in σ-transformed

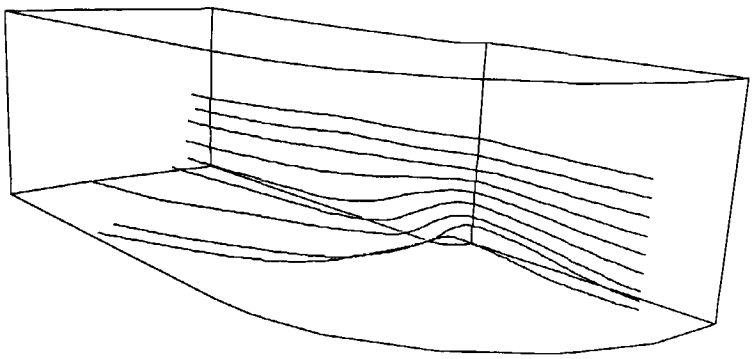


Figure 3.24: Blunt fin with particle paths released from a rake

grids, this 5-decomposition cannot cope with the problems of strongly sheared cells in the vertical direction. In that case, a 6-decomposition should be used. This decomposition does not entail greater computational costs.

For particle tracing in unstructured grids, extra connectivity information must be added to enable the particle tracing algorithm to traverse the grid. This connectivity information comprises a list of cell faces and the cells that share them, such that cell neighbours may be found. Although deriving this information is computationally expensive, it needs only be done once for each dataset, after which it can be reused in all visualizations. In this context, a standard representation for adjacency information would be useful.

Finally, we have shown examples of particle tracing in each grid type, using hydrodynamic datasets.

Chapter 4

Vortex detection

西嶽崢嶸何壯哉
黃河如絲天際來
黃河萬里觸山動
盤渦轂轉秦地雷[1]

Vortices are important features in many types of flow research, and they are studied for theoretical and practical purposes. In fundamental flow research, the evolution of vortices is of great importance. In engineering applications, such as machinery design and hydraulics, vortices can either be desirable or undesirable, and designs are optimized to prevent or to promote the occurrence of vortices. Visualization of vortices is therefore important for understanding the underlying phenomena, and also for simulating and modifying designs. Previous applications of vortex detection and visualization have been described in oceanography [Zhu & Moorhead, 1995], aerodynamics [Kenwright & Haimes, 1997], and turbomachinery design [Roth & Peikert, 1996].

Informally, a vortex is defined as a swirling flow pattern which will often behave as a coherent structure in time-dependent flows. In [Robinson, 1991], it was described as:

> *"A vortex exists when instantaneous streamlines mapped onto a plane normal to the vortex core exhibit a roughly circular or spiral pattern, when viewed from a frame of reference moving with the center of the vortex core."*

A formal definition of a vortex cannot be given. Although in fluid dynamics research, several criteria have been developed for their detection, the essential characteristics are hard to formalize, and none of the existing criteria is entirely satisfactory [Banks & Singer, 1994; Roth & Peikert, 1996].

This chapter presents vortex detection criteria of two types: physical criteria and geometric criteria. Physical criteria are covered in Section 4.1. One special type of physical criteria, critical points, is covered in Section 4.2. Next, two examples of a geometric criteria are described: one is based on curvature centres, for which the basic principles are described in Section 4.3, and for which enhancements are given in Section 4.4. Another one, described in Section 4.5, is based on the circular shape of streamlines. Finally, Section 4.6 summarizes the findings and gives some conclusions.

[1] How mighty the Western Hua Mountain stands tall! // The Yellow River comes like a thread from the horizon // The Yellow River hits the mountains for 10,000 miles // Vortices spinning about their axes thunder through Qin. (Li Bai)

4.1 Physical vortex detection criteria

The first category of vortex detection criteria is based on physical quantities that can be determined at any point of a flow field. These criteria consist of a physical quantity and a range, e.g. $Q > 0$ for some quantity Q. The criteria are based on assumptions about the characteristics of the flow patterns in an infinitely small zone around a point. The quantities are usually scalars, although they are sometimes derived from vector or tensor quantities, such as velocity and quantities derived from the velocity. With these quantities, different kinds of criteria may be derived. This section gives a few examples, but does not intend to be complete or exhaustive. Other brief surveys may be found in [Banks & Singer, 1994; Roth & Peikert, 1996; Portela, 1997]. For a comprehensive text book on fluid mechanics, see [Batchelor, 1967–1994].

4.1.1 Criteria descriptions

One frequently used criterion for detecting vortices is *low pressure*. In a free flow without obstacles, a pressure gradient with low pressure at a vortex core can drive the rotational motion. Unfortunately, in flows governed by large pressure gradients, due to obstacles or walls, these pressure gradients will dominate the much smaller pressure gradients associated with vortices or rotational motion. In [Portela, 1997], it is even shown that there are vortices that have a pressure *maximum* at the vortex core. So this criterion is not always sufficient.

Another group of criteria used for identifying vortices is derived from the velocity field. An important criterion is *high vorticity magnitude*. Vorticity is a vector quantity defined as the curl of the velocity field: $\boldsymbol{\omega} = \nabla \times \mathbf{v}$. Since vorticity is proportional to the angular velocity of a fluid particle, it is easy to draw the conclusion that where there is a vortex, there is rotation, so the vorticity magnitude should be high. However, the opposite is not always true: the presence of high vorticity does not guarantee the presence of a vortex. The simplest example is a shear flow, where there is high vorticity at every point, yet there are no vortices. Another criterion derived from the velocity field is *high normalized helicity* (H_n), a scalar quantity defined as $H_n = \frac{\mathbf{v} \cdot \boldsymbol{\omega}}{|\mathbf{v}| \cdot |\boldsymbol{\omega}|}$ [Buning, 1989]. This may also be regarded as the cosine of the angle between the velocity and the vorticity. H_n reaches its maximum when $\mathbf{v}$ is parallel to $\boldsymbol{\omega}$. This is supposed to occur at vortex cores, but sometimes the helicity criterion fails to detect part of the vortex core [Banks & Singer, 1994].

Yet other criteria, which are also derived from the velocity field, use the properties of the velocity gradient, or rate-of-deformation tensor $\nabla \mathbf{v}$. In particular, complex or imaginary eigenvalues of this tensor indicate rotational movement. In [Chong *et al.*, 1990], the authors suggest a definition of vortex cores as regions where $\nabla \mathbf{v}$ has *complex eigenvalues*, so that the gradient tensor is dominated by the rotation component. In [Helman & Hesselink, 1991], a special case is used, by not checking for complex eigenvalues of $\nabla \mathbf{v}$ at all (grid) points of the field, but only around critical points, i.e. where $\mathbf{v} = \mathbf{0}$. However, in many flows resulting from simulations there are much fewer critical points than vortices [Roth & Peikert, 1996], or they are at different locations than

the vortices [Banks & Singer, 1994]. In [Jeong & Hussain, 1995], vortices are defined as regions where $\lambda_2 < 0$. Here, λ_2 is defined as the second-largest eigenvalue of the tensor $S^2 + \Omega^2$, where $S = \frac{1}{2}(\nabla \mathbf{v} + (\nabla \mathbf{v})^T)$ and $\Omega = \frac{1}{2}(\nabla \mathbf{v} - (\nabla \mathbf{v})^T)$ are the symmetric and anti-symmetric parts of the velocity-gradient tensor $\nabla \mathbf{v}$. This criterion is based upon the Navier-Stokes equations, and boils down to finding those regions of low pressure which are caused only by swirling motion, rather than by strain or viscosity. However, since it is also based on pressure, this criterion has the same drawbacks as the low-pressure criterion.

Since single scalar quantities have often not proven satisfactory, some researchers have tried combinations of multiple scalar criteria. In [Hunt *et al.*, 1988], a combined criterion was used of low pressure and a positive second invariant of $\nabla \mathbf{v}$, while [Banks & Singer, 1994] used a criterion of low pressure and high vorticity magnitude. However, the latter only used this criterion as a heuristic method to find seed points, for applying a more algorithmic method for tracking vortex cores. These purely algorithmic methods also include the algorithm proposed in [Villasenor & Vincent, 1992] (see Section 2.5).

4.1.2 Example

We now show some examples of physical criteria applied to a single data set. The goal is to verify how well the physical criteria are able to detect this pattern. The data set we use is the result of a simulation performed at the NASA-Ames Research Center of a laminar flow past a tapered cylinder [Jespersen & Levit, 1991]. This tapered cylinder has a variable radius along the length axis, which influences the vortex shedding frequency along the cylinder. The grid used is a structured, cylindrical grid with $64 \times 64 \times 32$ nodes, each of which contains density, x, y, z-momentum, and stagnation. The simulation is time-dependent, but we use only one time step.

Figure 4.1 shows a global view of the data set, with the cylinder in the centre, the bottom plane of the cylindrical grid, and a rectangular xy-plane at $z = 16$, in which streamlines are released. The cylinder is located at the origin and has a radius of approximately $r = 0.5$.

Figure 4.2a shows a slice at $z = 18$, coloured with pressure. The arrow plot of the vector field shows a vortex behind the 'lower half' of the cylinder. It can be seen that the global pressure minima are located close to the cylinder surface, on the top and bottom sides (near $(x, y) = (0, \pm 0.5)$, although there are no vortices there. There is also a region of slightly lower pressure, which partly corresponds to the shape of the vortex to be found, but not completely. Therefore, pressure is not a sufficient quantity for detecting this vortex.

Figure 4.2b shows the same slice coloured with vorticity magnitude ω. It can be seen that the maxima of the vorticity magnitude are located near the cylinder surface again. There are also regions with lower values which have shapes similar to the Von Kármán vortices well-known from fluid dynamics [Batchelor, 1967–1994]. However, these vorticity regions correspond even less with the vortex shape than the pres-

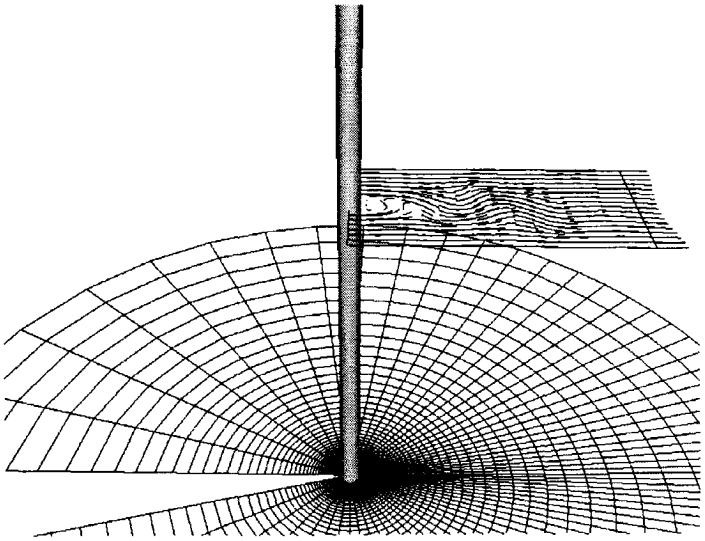


Figure 4.1: Global view of the flow past tapered cylinder, with the cylinder, the bottom grid slice, and a streamline plane. The flow is from left to right.

sure region. Therefore, vorticity magnitude is also not a satisfactory quantity for detecting this vortex.

Figure 4.3a (see also Colour Plate 1) shows the slice coloured with normalized helicity H_n. High values of H_n are located in numerous regions. It can be seen that there is a local maximum at the location of the vortex core. So in this case, high normalized helicity could be a suitable criterion for finding the core of this vortex, although further selection would be necessary to filter out spurious maxima not belonging to vortices.

To find the boundary of this vortex, we would like to use a contour line of a slightly lower value of the normalized helicity. Unfortunately, the contour lines of the helicity field in the region around the vortex core do not correspond to the streamlines in that region. Therefore, this criterion would not be suitable for detecting the boundary of this vortex.

Figure 4.3b (see also Colour Plate 2) shows the slice coloured with λ_2, and white contour lines of $\lambda_2 = 0$. The points where $\lambda_2 < 0$, which according to the criterion should constitute vortices, are inside these curves. It can be seen that the shape of these curves is also similar to Von Kármán vortices. One of the large curves, the one in the lower half of the image, indeed contains the vortex as indicated by the arrow plot. Unfortunately, the other large $\lambda_2 = 0$ curve, in the upper half of the image, does not correspond to any vortex in the arrow plot at all. So, the λ_2 criterion is suitable as a pre-selection technique, but extra verification is necessary.

This example illustrates how physical criteria often fail to locate vortices. In some cases, certain criteria give an indication of where vortices are located, but they seldom

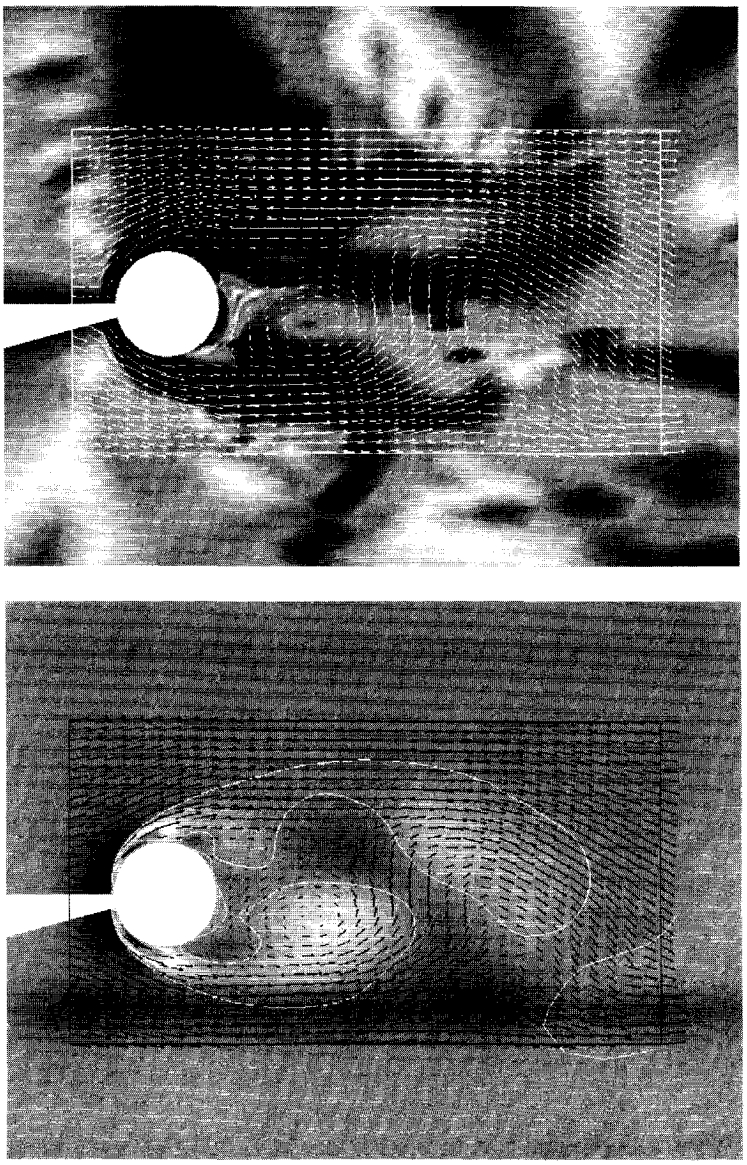


Figure 4.3: Tapered cylinder slice coloured with (a) normalized helicity (b) λ_2. On the white curves, $\lambda_2 = 0$.

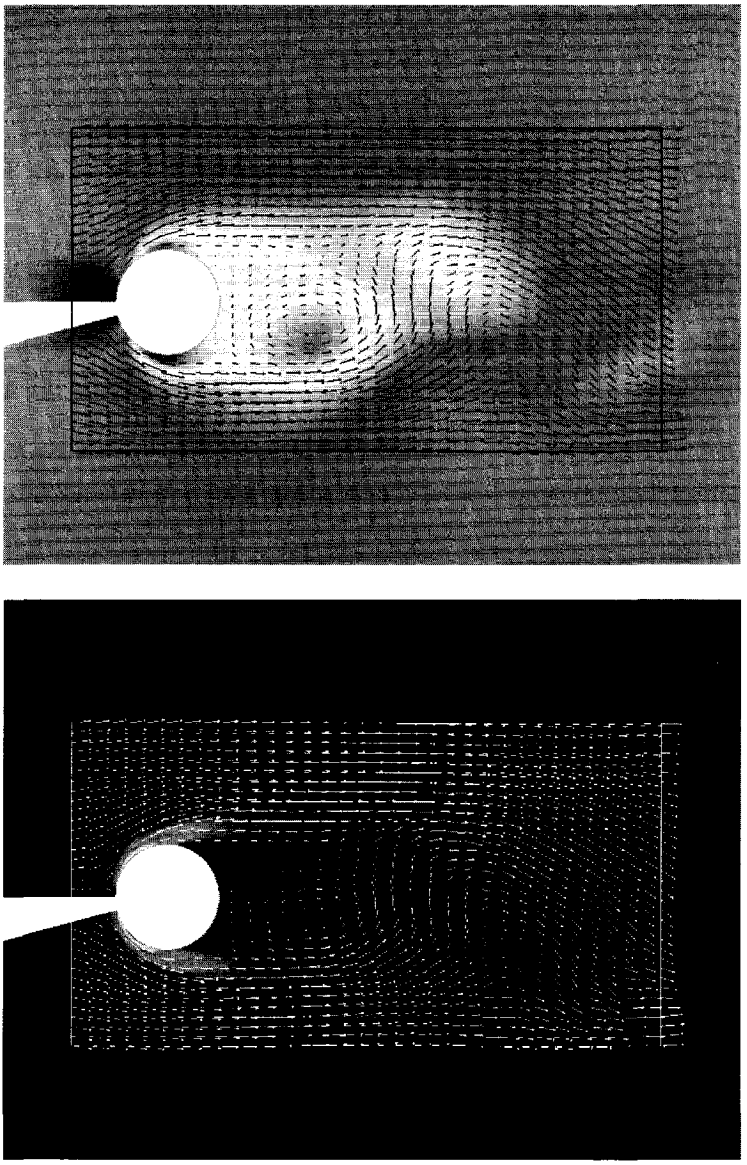


Figure 4.2: Tapered cylinder slice coloured with pressure (top) and vorticity magnitude (bottom).

correspond *exactly* with vortices seen in the arrow plots. None of the criteria works in all cases and extra verification is always necessary. We attribute this weakness to the fact that these criteria are based on point samples, while vortices are obviously regional phenomena. Apparently, it is not possible to extend the properties of infinitesimal regions to larger regions.

4.2 Critical points for vortex detection

It has been reported that vortex cores can be located by calculating critical points of the velocity field, i.e. points of zero velocity [Kenwright & Haimes, 1997], [Zhu & Moorhead, 1995]. Spiralling or circular flow is then indicated by complex eigenvalues of the velocity gradient tensor at the critical points. We have investigated this, too.

Critical points in 2D can be calculated as follows. Given a quadrilateral cell consisting of four nodes where the velocity x-components u are $(u_{00}, u_{01}, u_{10}, u_{11})$ and the velocity y-components are $(v_{00}, v_{01}, v_{10}, v_{11})$. Then, the interpolated velocity at point (α, β) is given by:

$$u(\alpha,\beta) = (1-\alpha)(1-\beta)u_{00} + \alpha(1-\beta)u_{10} + \beta(1-\alpha)u_{01} + \alpha\beta u_{11} \quad (4.1)$$

$$v(\alpha,\beta) = (1-\alpha)(1-\beta)v_{00} + \alpha(1-\beta)v_{10} + \beta(1-\alpha)v_{01} + \alpha\beta v_{11} \quad (4.2)$$

We want to determine the coordinates (α, β) where $u(\alpha, \beta) = v(\alpha, \beta) = 0$. Regrouping the above terms for α and β, and requiring both velocity-components to be zero, we obtain:

$$u_{00} + (u_{10} - u_{00})\alpha + (u_{01} - u_{00})\beta + (u_{00} + u_{11} - u_{10} - u_{01})\alpha\beta = 0 \quad (4.3)$$

$$v_{00} + (v_{10} - v_{00})\alpha + (v_{01} - v_{00})\beta + (v_{00} + v_{11} - v_{10} - v_{01})\alpha\beta = 0 \quad (4.4)$$

or, by substituting letters for the coefficients:

$$K + L\alpha + M\beta + N\alpha\beta = 0 \quad (4.5)$$

$$P + Q\alpha + R\beta + S\alpha\beta = 0 \quad (4.6)$$

from which it follows that:

$$\alpha = -(K + M\beta)/(L + N\beta) \quad (4.7)$$

$$\beta = -(P + Q\alpha)/(R + S\alpha) \quad (4.8)$$

By substituting (4.7) in (4.8), we obtain:

$$(NR + MS)\beta^2 + (NP + LR - KS - MQ)\beta + (LP - KQ) = 0 \quad (4.9)$$

This gives at most two solutions for β, from which α can be obtained through back-substitution in (4.7). The real solutions for which $0 \leq \alpha, \beta \leq 1$ are valid critical points.

Then, the presence of a vortex core may be indicated by the type of eigenvalues of the velocity gradient $\nabla \mathbf{v}$ at the critical points. If the eigenvalues are complex, this

indicates rotational or spiralling flow around the critical point [Helman & Hesselink, 1991].

However, there are a few caveats: just like the physical quantities of Section 4.1, this vortex detection method is based on samples in an infinitesimal region (in which the velocity gradient is calculated). Therefore, it might not work well in regions with large vortices. Also, since the critical points are defined as points of zero velocity, they are sensitive to noise: adding a small constant to the entire field moves the location of all the critical points. Thirdly, this method does not allow to calculate vortex attributes, as opposed to the winding-angle method described in Section 4.5.

4.3 The curvature centre method

This section describes a *geometric method* for vortex detection in 2D. Geometric methods are exclusively based on the geometric properties of flow curves, not on any physical quantities. More specifically, they try to detect vortices by looking for loops in flow curves. While there is no generally accepted formal definition of a vortex, it is generally agreed upon that vortices consist of more or less circular flow patterns, reflected in circular flow curves.

4.3.1 Method description

The curvature centre method works by sampling the field at many points, typically at all grid nodes. For each sample point, the *curvature centre* is determined, which is the centre of the osculating circle of the streamline through that point [Farin, 1990]. In vortical regions of the field, characterized by (almost) circular pathlines, the curvature centres should accumulate in a small area [de Leeuw & Post, 1995]. Figure 4.4a shows an example of how the samples taken on a perfectly circular streamline all project to the same curvature centre. In non-vortical regions of the field, characterized by non-circular pathlines, the curvature centres will be randomly scattered, as in Figure 4.4b.

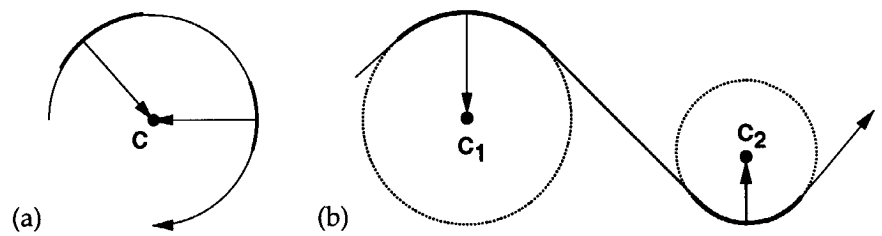


Figure 4.4: (a) Circular streamline with coinciding curvature centres c and (b) Non-circular streamline with scattered curvature centres.

By calculating curvature centres for many sample points, we obtain a set of curvature centre points, which are accumulated into a new grid. The number of curvature

centres in each cell constitutes a new scalar field which we call the *Curvature Centre Density* (CCD) field. Vortex cores could then be located in cells with high values of the CCD field.

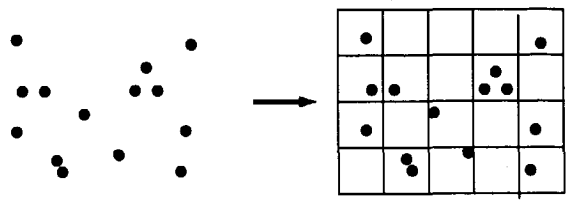


Figure 4.5: Curvature centre points are accumulated into a new grid, resulting in the Curvature Centre Density (CCD) scalar field.

The computational details for calculating curvature centres are as follows. For a position $\mathbf{p}$, the curvature centre $\mathbf{c}(\mathbf{p})$ is determined by adding to that position the principal normal $\mathbf{N}$ to the streamline scaled by the radius of the curvature κ at that position [Farin, 1990]:

$$\mathbf{c}(\mathbf{p}) = \mathbf{p} + \frac{\mathbf{N}}{\kappa} \tag{4.10}$$

It can be shown that $\frac{\mathbf{N}}{\kappa}$ is equivalent to the centripetal acceleration $\mathbf{a}_c$, so (4.10) can be simplified to:

$$\mathbf{c}(\mathbf{p}) = \mathbf{p} + \mathbf{a}_c \tag{4.11}$$

which is more practical, because the centripetal acceleration can be determined as the directional derivative of $\mathbf{v}$:

$$\mathbf{a}_c = (\nabla \mathbf{v}) \cdot \mathbf{v} \tag{4.12}$$

Each curvature centre in the CCD field has an associated *weight* factor w which is the velocity magnitude $|\mathbf{v}|$ (see Section 4.4 for other weight factors).

So far, this method has concentrated on detecting vortex cores. Usually, we are also interested in vortex boundaries. One way to detect the vortex boundaries could be based on so-called *basins* of the highest peaks of the CCD field. A basin is defined as the neighbourhood of a vortex core where the streamlines, or velocity vectors, contribute to that core. This is the case when the curvature centres of those streamlines are part of that core. A basin can be determined by calculating the curvature centre for each grid node of the velocity field, and checking if the centre is part of a high peak (vortex core) in the CCD field. If it does, then the grid node is part of the basin of that vortex. Which peaks are considered high is defined using a user-specified threshold value. In this way, a Boolean field is obtained which tells for each grid node whether it belongs to a strong vortex. However, it does not tell which grid nodes belong to which vortices. For this purpose, a clustering algorithm can be applied, to recognize a limited number of vortex basins and associate the closest points of the Boolean field with them.

4.3.2 Example

As an example, we use a data set calculated in a numerical simulation of the surface layer of the Pacific Ocean [Zhu & Moorhead, 1995]. The simulation employs the US Navy layered ocean model, a tool used by the Naval Oceanographic and Atmospheric Research Laboratory to assist in ocean prediction. The data set is defined on a uniform grid of 117 × 84 nodes, at each of which a 2D velocity vector is defined, and an array to specify dry (land) points. For more information on this simulation, see Section 6.1. Figure 4.6 shows the global flow patterns using streamlines released from every grid point.

Figure 4.6: Pacific Ocean with streamlines

Figure 4.7 shows the curvature centre density field derived from the velocity data. The 2D scalar field has been rendered as a height field, to make the peaks easier to identify.

Figure 4.8 shows the CCD field combined with streamlines, to show how well the peaks in the CCD field correspond to rotational patterns in the streamlines. The CCD field has been thresholded to find the peaks where $CCD \geq 0.3 \max(CCD)$. See also

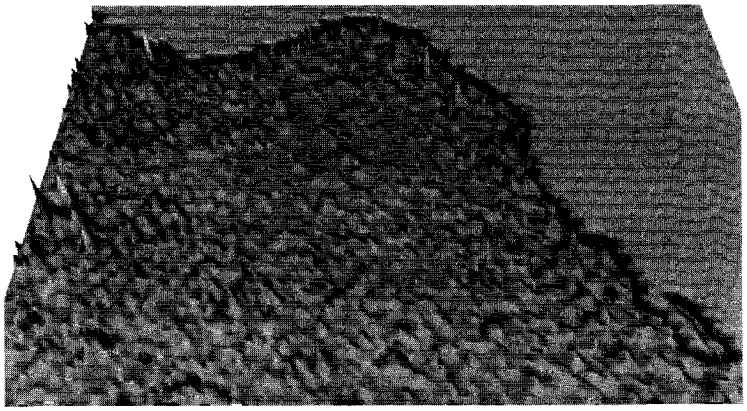


Figure 4.7: Pacific Ocean with CCD field

the bottom image on the back cover where the streamlines have been coloured with the velocity magnitude.

While the peaks of the CCD field correspond to the vortex cores, the basins of the CCD field correspond to the vortex boundaries. Figure 4.9 shows the basins contributing to the same CCD peaks as in the previous figure ($CCD \geq 0.3 \max(CCD)$).
From these figures, a number of problems become clear. The curvature centre density field has:

1. false peaks: many peaks do not correspond to vortices ('false positives')
2. missing peaks: some obvious vortices do not show up as peaks ('false negatives').
3. low peaks: the number of samples per peak is too low.
4. noise: the difference in peak height between vortices and non-vortices is too small (low signal/noise ratio).

Another problem, which is not visible from the figure, is the sensitivity of the result to the cell size in the output grid. Depending on the cell size, certain peaks may appear or disappear. This is a discretization effect, related to the low number of samples (problem 3). These problems call for enhancements to the basic technique, which will be covered in the next section.

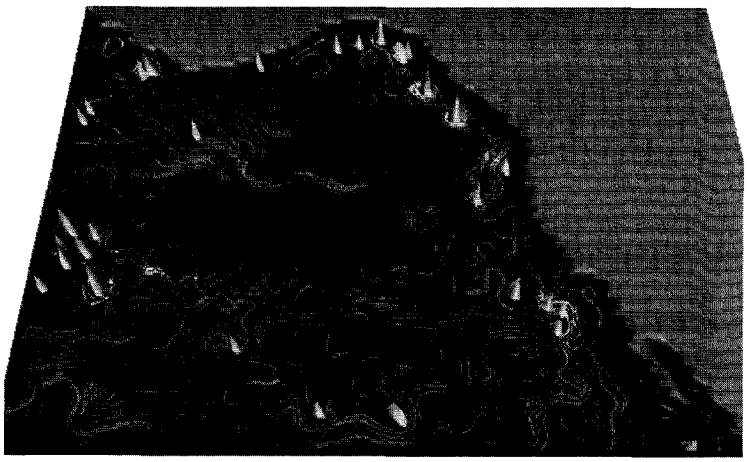


Figure 4.8: Pacific Ocean with streamlines and peaks of $CCD \geq 0.3 \max(CCD)$

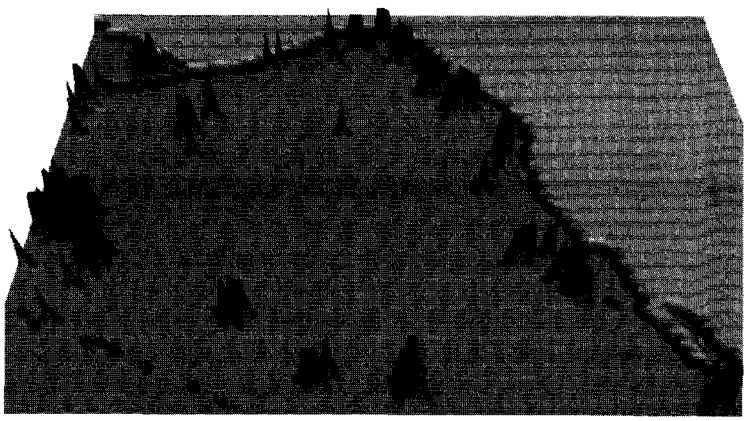


Figure 4.9: Pacific Ocean with basins of $CCD \geq 0.3 \max(CCD)$

4.4 The enhanced curvature centre method

The curvature centre method as described in the previous section suffers from a number of problems. Possible solutions to these problems are:

1. thresholding: selecting only values higher than a given threshold may filter out unwanted low peaks.
2. filtering: applying a low-pass filter to the field may eliminate noisy peaks.
3. supersampling: increasing the number of samples may enhance the true peaks, while leaving the false peaks low.
4. signed weights: expressing the rotation direction as a sign may eliminate pseudo-vortices caused by streamlines rotating in opposite directions.
5. other weights: by default, curvature centres are weighted by the velocity magnitude. Including other factors in the weight may give different peaks.

These are described below.

4.4.1 Enhancements

1. *Thresholding* may be applied to alleviate the problem of too many peaks in the density field. The threshold operator discards those points in the CCD field which are lower than a certain threshold value. The challenge is to find a good threshold value which clearly distinguishes the true peaks from the false peaks.

2. An alternative to thresholding is *filtering*. A low-pass filter may be applied to smooth the field by filtering out noise. We could use a standard Gaussian filter with varying kernel sizes, depending on how much smoothing we want to perform. Although the goal of filtering is the same as that of thresholding, i.e. to find the highest peaks in the field, the approach is different. Whereas thresholding only selects data, filtering changes data.

3. *Supersampling* may be applied to alleviate the problem that the peaks are too low, resulting from a low sampling rate. By specifying a supersampling rate of r, instead of taking only the values at the grid points, r samples are taken per grid cell in each direction, which is r^2 samples per 2D cell.

4. *Signed weights* may be used to eliminate 'pseudo-vortices'. These are pairs of pathlines having a common curvature centre, but flowing in opposite directions (see Figure 4.10). If rotation directions are not taken into account, this may lead to false peaks.

Pseudo-vortices may be eliminated by using signed weights: if a streamline is rotating in one direction, say counterclockwise, a positive weight is added to the curvature centre field; if it is rotating in the other direction, a negative weight is added. This results in a CCD field with positive peaks for vortices with counterclockwise rotation, negative peaks for vortices with clockwise rotation, and no peaks for pseudo-vortices, because weights with opposite signs cancel out.

In 2D, the rotation direction may be determined by looking at the cross product of the velocity $\mathbf{v}$ and the centripetal acceleration $\mathbf{a}$: $s = (\mathbf{v} \times \mathbf{a}_c)$. In 2D, this cross prod-

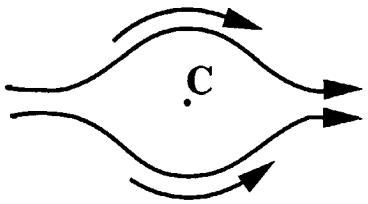


Figure 4.10: Pseudo-vortices, caused by streamlines having common curvature centre

uct is a scalar, with $s < 0$ indicating clockwise rotation, and $s > 0$ counterclockwise rotation.

5. *other weights*: There are many possible choices for the weight w used in calculating the curvature centre density. The weight was initially chosen to be $w = 1$, in other words, every curvature centre was given the same weight. When this turned out to give inconclusive results, the weight was changed to the velocity magnitude $|\mathbf{v}|$.

Other choices for the weight w may take into account not only the velocity magnitude $|\mathbf{v}|$, but also the distance r between the streamline point $\mathbf{p}$ and the curvature centre C. The rationale for considering the velocity magnitude, is that strong vortices should give proportionally higher peaks than weak vortices. This would lead to a factor of $|\mathbf{v}|$ in the weight w. The rationale for considering the distance, is that the influence of a rotating streamline (rotating mass) on a vortex core should decrease as it lies farther from the core (at distance r). This would lead to a factor of $1/r$ in the weight w. This leads to the weights listed in Table 4.1.

take into account:	$\|\mathbf{v}\|$	
r	no	yes
no	1	$\|\mathbf{v}\|$
yes	1/r	$\|\mathbf{v}\|/r$

Table 4.1: Various weighting schemes

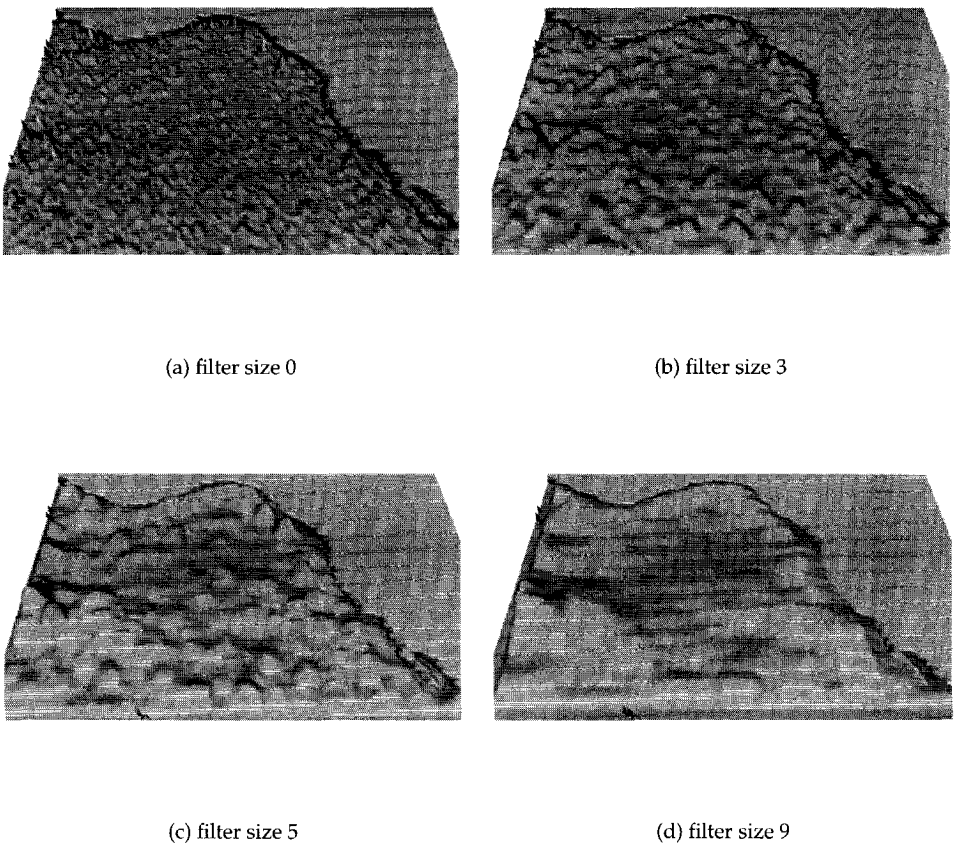
4.4.2 Example

We use the same example as in the previous section to examine the effect of the proposed enhancements.

1. The effect of *thresholding* is as expected. Higher values of the threshold value filter out more peaks and leave fewer peaks, lower values leave more peaks. Unfortunately, it was difficult to find a good threshold level which clearly distinguishes between true and false peaks. Therefore, thresholding did not give sufficient improvement.

2. The effect of *filtering* using a Gaussian filter of various kernel sizes is shown in Figure 4.11. It can clearly be seen that filtering has little effect for the smaller filter sizes,

as many false peaks remain present after filtering. Filtering has too much effect for the larger filter sizes, as not only the false peaks are smoothed out, but also the true peaks.



(a) filter size 0

(b) filter size 3

(c) filter size 5

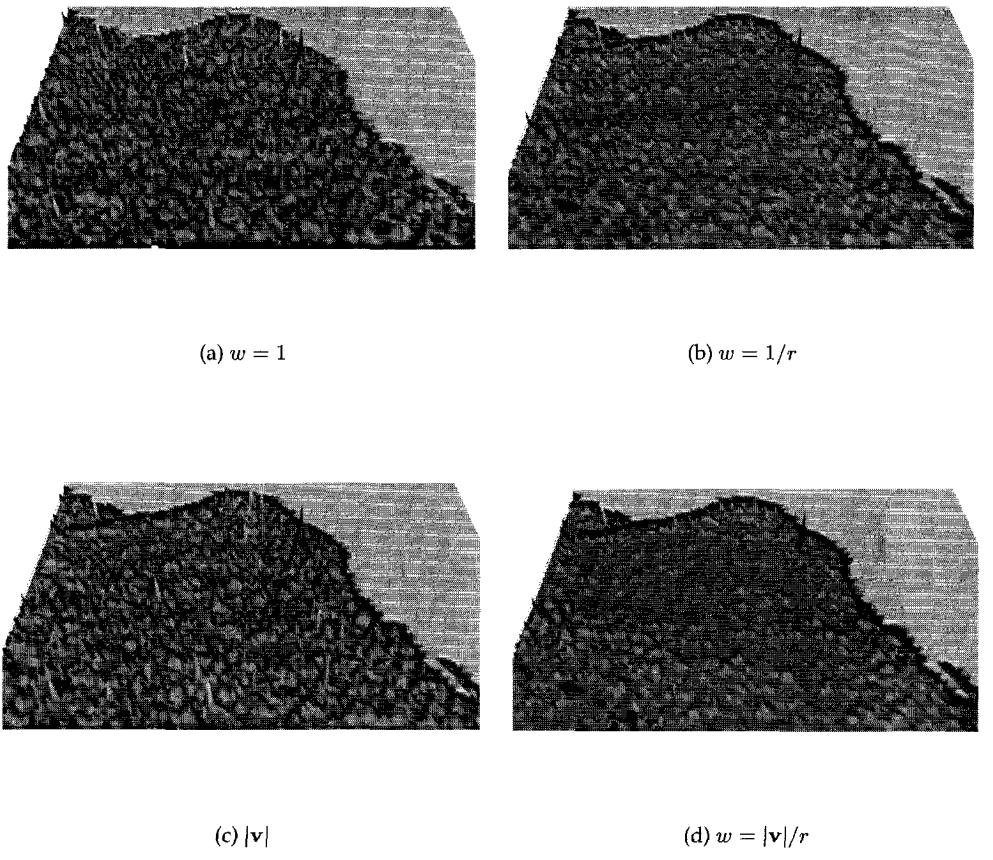
(d) filter size 9

Figure 4.11: CCD field filtered with various filter sizes

3./4. The effects of the next two enhancements, *supersampling* and *signed weights*, turned out to be negligible, and are not shown here.

5. The effect of *other weighting schemes* is shown in Figure 4.12. We see that the uniform weighting scheme $w = 1$ is worst, because there are many high peaks, with no distinction between real and false peaks. We also see that the schemes that contain a factor $|\mathbf{v}|$ are better than the ones that do not, and that the schemes that contain factor $1/r$ are also better than the ones that do not. Therefore, it is logical that $w = |\mathbf{v}|/r$ is the best choice, which can also be seen in the figure. Apparently, both the velocity magnitude and the distance to the vortex core should be taken into account in the weight

w. Unfortunately, this scheme is only relatively good, compared to the others. In the absolute sense, even the best scheme still does not make a clear distinction between vortices and "background", which is what we want.



(a) $w = 1$

(b) $w = 1/r$

(c) $|\mathbf{v}|$

(d) $w = |\mathbf{v}|/r$

Figure 4.12: Pacific Ocean with density field with various weighting schemes

4.4.3 Discussion

Some of the enhancements in this section give a slight improvement over the basic technique. The main problem of the curvature centre method seems to be that it is sensitive to not perfectly circular streamlines. If the streamlines are not perfectly circular but elliptic, this has a large influence on the CCD field. Elliptic shapes are often

caused by shear flow or interacting vortices. This is illustrated by two analytical flows: a circular and an elliptic flow.

circular flow

The ideal test case for the curvature centres method is a circular flow. Since the streamlines are perfect concentric circles, all the curvature centres should coincide. Let the velocity field $\mathbf{v}(x,y) = (v_x(x,y), v_y(x,y))$ of the circular flow be defined by

$$\mathbf{v}(x,y) = (y,-x) \tag{4.13}$$

Figure 4.13a depicts streamlines of this velocity field. The centripetal acceleration $\mathbf{a}_c(x,y) = (a_x, a_y)$ is given by

$$\begin{aligned} \mathbf{a}_c(x,y) &= v_x \frac{\partial \mathbf{v}}{\partial x} + v_y \frac{\partial \mathbf{v}}{\partial y} && (4.14)\\ &= y \begin{pmatrix} 0 \\ -1 \end{pmatrix} - x \begin{pmatrix} 1 \\ 0 \end{pmatrix} = (-x,-y) && (4.15) \end{aligned}$$

which results in the curvature centre $\mathbf{c}(\mathbf{p})$:

$$\mathbf{c}(\mathbf{p}) = \mathbf{p} + \mathbf{a}_c = (x,y) + (-x,-y) = \mathbf{0} \tag{4.16}$$

which proves that for perfect, circular streamlines, the curvature centres coincide in the origin (0,0), as Figure 4.13b shows.

elliptic flow

In the case that the streamlines are not perfect circles, but deformed to ellipses, the velocity field is given by:

$$\mathbf{v}(x,y) = (-ay, bx) \tag{4.17}$$

with $a, b > 0$ and $a \neq b$. Figure 4.13c shows an example with $a = 2, b = 1$.

Then, the centripetal acceleration $\mathbf{a}_c = (a_x, a_y)$ is given by

$$\begin{aligned} \mathbf{a}_c(x,y) &= v_x \frac{\partial \mathbf{v}}{\partial x} + v_y \frac{\partial \mathbf{v}}{\partial y} && (4.18)\\ &= -ay \begin{pmatrix} 0 \\ b \end{pmatrix} + bx \begin{pmatrix} -a \\ 0 \end{pmatrix} && (4.19)\\ &= \begin{pmatrix} -abx \\ -aby \end{pmatrix} = -ab \begin{pmatrix} x \\ y \end{pmatrix} && (4.20) \end{aligned}$$

which results in the curvature centres $\mathbf{c}(\mathbf{p})$:

$$\mathbf{c}(\mathbf{p}) = \mathbf{p} + \mathbf{a}_c = (x,y) - ab(x,y) = (1-ab)(x,y) \tag{4.21}$$

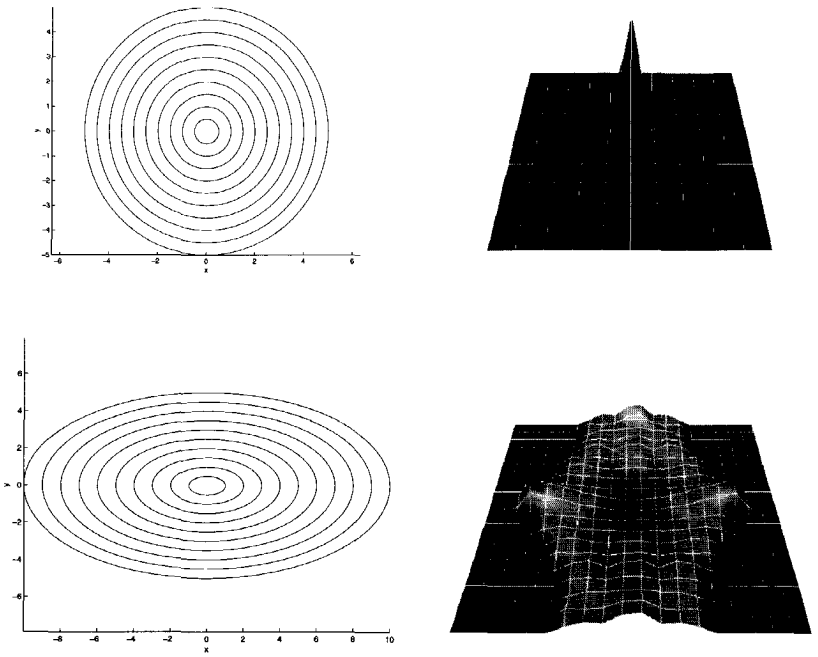


Figure 4.13: In circular flow (top), there is a peak in the CCD field. In elliptic flow (bottom), the peak in the CCD field is spread out.

Figure 4.13d shows the curvature centre density field for this flow. It can be seen that the peak is spread out into a flatter cross-like shape. This is a serious problem, because it means that streamlines which are not perfectly circular will not lead to high peaks in the curvature centre density field, even though they could be part of vortices.

It can be concluded that some of the enhancements in this section give a slight improvement over the basic technique, but not enough. In the next section, we examine a very different type of geometric technique.

4.5 The winding-angle method

The curvature centre methods described in the previous sections are not satisfactory, because they depend too much on the perfect circularity of flow curves. This section describes another geometric method, which does not rely on the exact shape of a flow curve, but on its rotation direction: the curve must make a loop. We call this method the *winding-angle* method [Sadarjoen *et al.*, 1998b; Sadarjoen & Post, 1999a].

Basically, the winding-angle method keeps track of the change of direction along

the curve, based on a winding-angle concept. This is a simpler version of the winding-angle used in [Portela, 1997].

4.5.1 Method description

The method first selects curves (streamlines or pathlines) which make a winding (looping) motion in one direction, i.e. clockwise or counter-clockwise. The speed or the exact shape of the curve is not important, only the looping shape. Next, the selected streamlines are used for determining numeric vortex attributes. These numeric attributes can either be used for further analysis, or they can be visualized using icons.

One advantage of this method is that it can detect vortices consisting of streamlines that are not perfect circles, but e.g. elliptical. Another advantage is that this technique is not only a visual, qualitative selection technique, but also a quantitative feature extraction technique. A limitation is that the method currently only works in 2D.
The vortex extraction process consists of the following stages:

- selection
- clustering
- quantification
- visualization mapping

The first stage, *selection*, selects those streamlines that have:

1. a high winding-angle
2. an end point close to the starting point

We define the *winding-angle* of a streamline as the cumulative change of direction of the streamline segments. Let S_i be a 2D streamline, consisting of points $P_{i,j}$ and line segments $(P_{i,j}, P_{i,j+1})$, and let $\angle(P_{i,j-1}, P_{i,j}, P_{i,j+1}))$ denote the angle between line segments $(P_{i,j-1}, P_{i,j})$ and $(P_{i,j}, P_{i,j+1})$. Then, the *winding-angle* $\alpha_{w,i}$ of streamline S_i is given by:

$$\alpha_w = \sum_{j=2}^{N-1} \angle(P_{i,j-1}, P_{i,j}, P_{i,j+1}) \tag{4.22}$$

This is illustrated in Figure 4.14. We use signed angles, with a positive sign for a counterclockwise rotation, and a negative sign for a clockwise rotation. Obviously, $\alpha_{w,i} = \pm 2\pi$ for a fully closed curve; lower values may be used to find winding streamlines which do not make a full revolution.

The definition of a 'high' winding-angle depends on the case. In cases where all the streamlines have at least one full winding, a threshold of 2π can be applied. In other cases, when there are many slowly rotating vortices, a threshold of 1.5π may be sufficient.

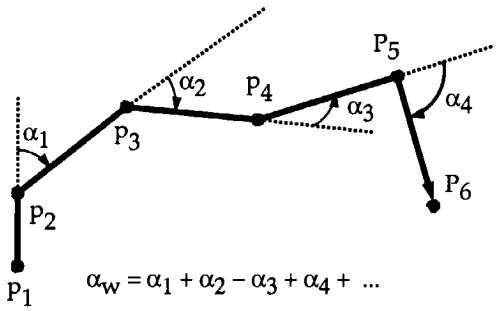


Figure 4.14: The winding-angle α_w is the sum of the angles between the edges.

The definition of 'close' uses both an absolute and a relative distance between the initial and the final point of the pathline. The relative distance is defined in terms of the estimated radius r_{est}, which is determined as the average distance between all the points and their geometric mean.

The purpose of *clustering* is to group those streamlines together which belong to the same vortex. Rather than clustering streamlines, it is easier to cluster points. To this end, each streamline is mapped to a point by determining the centre point, or geometric mean, of all the sample points on that streamline. These centre points are then clustered as follows. The first cluster is formed by the first point. For each subsequent point, it is determined which previous cluster lies closest. If the point is not within a predetermined radius of all the existing clusters, it constitutes a new cluster. In this way, the selected streamlines are combined into a distinct number of groups. Streamlines of the same group are deemed to rotate around the same core, and to be part of the same vortex.

Once the streamlines have been clustered, *quantification* of the vortices is performed by calculating numeric attributes of the corresponding streamline clusters. We approximate the shape of the vortices by *ellipses*. Fitting an ellipse to a set of points is done by calculating statistical attributes, such as mean, variance, and covariance, of the points [van Walsum *et al.*, 1996; Reinders *et al.*, 1997]. In addition, we calculate specific vortex attributes, such as rotation direction and angular velocity. We denote the number of points on a streamline S_i as $|S_i|$, a cluster of streamlines as $C_k = \{S_{k,1}, S_{k,2}, \ldots\}$, where $S_{k,l}$ is streamline #l in cluster #k, the number of streamlines in that cluster as $|C_k|$, and all the points on all the streamlines in it as $\Psi(C_k)$. Now, we can calculate the following attributes for each vortex:

- streamline centre: $\bar{S}_i = \frac{1}{|S_i|}\sum_{j=1}^{|S_i|} P_{i,j}$
- cluster centre: $\bar{C}_k = \frac{1}{|C_k|}\sum_{l=1}^{|C_k|}(\bar{S}_{k,l})$
- cluster covariance: $M_k = cov(\Psi(C_k))$
- ellipse axis lengths: $\lambda_k = \mathrm{eig}(M_k)$
- ellipse axis directions: $\mathbf{d}_k = \mathrm{eigvec}(M_k)$

- vortex rotation direction: $d_k = sign(\alpha_{w,k})$
- vortex angular velocity: $\omega_k = \frac{1}{|C_k|\Delta t}\sum_{l=1}^{|C_k|}\alpha_{w,l}$

Visualization of the vortices can be accomplished by mapping their attributes to icons: the first three attributes are mapped to an ellipse. The angular velocity is visualized by adding wheel spokes to the ellipse, the number of which is made proportional to the angular velocity: fast rotation is suggested by many spokes, slow rotation by few. Finally, the rotation direction of a vortex is visualized by arrow heads.

4.5.2 Example

To test this method, we use the same flow past the tapered cylinder as in Section 4.1. First, we apply the winding-angle method to select streamlines from six slices in various z-planes, to show varying patterns. The selection criteria are: $\alpha_w \geq 1.5\pi$ and $d_{max,abs} \leq 0.5$.

Next, we apply clustering with maximum cluster radius 0.5, and quantification, which results in the numeric attributes listed in Table 4.2. This table reveals that for $z < 18$, there is only one counterclockwise vortex, for $18 \leq z \leq 20$, there are two vortices, one clockwise and one counterclockwise, and for $z = 21$, the counterclockwise vortex has disappeared, leaving only a large clockwise vortex.

z-plane	#vortices	max λ_{CW}	max λ_{CCW}
16	1	-	0.5305
17	1	-	0.5832
18	1	0.6216	0.2076
19	2	0.8446	0.4968
20	2	1.1138	0.6117
21	1	-	0.9370

Table 4.2: Statistics of the vortices in the flow past a tapered cylinder.

Figure 4.15 shows the visualization of these vortices. Grey streamlines show the global flow patterns, with the flow going from left to right past the cylinder, which is drawn as a semi-circle on the left. Selected streamlines are drawn in black. The ellipses approximate the shape and size of the vortices, with arrows indicating the rotation direction, and the number of spokes indicating the rotation speed (vortex strength): more spokes indicate faster rotation.

The middle image on the back cover shows a slightly different visualization of Figure 4.15e ($z = 20$), with different ellipse icons. These do not have the varying number of spokes, but the colour indicates the rotation direction: the red ellipse shows counterclockwise rotation, the green one clockwise rotation.

This example shows that this method produces better results than the physical criteria or the curvature centre method. One advantage is that it finds rotating structures

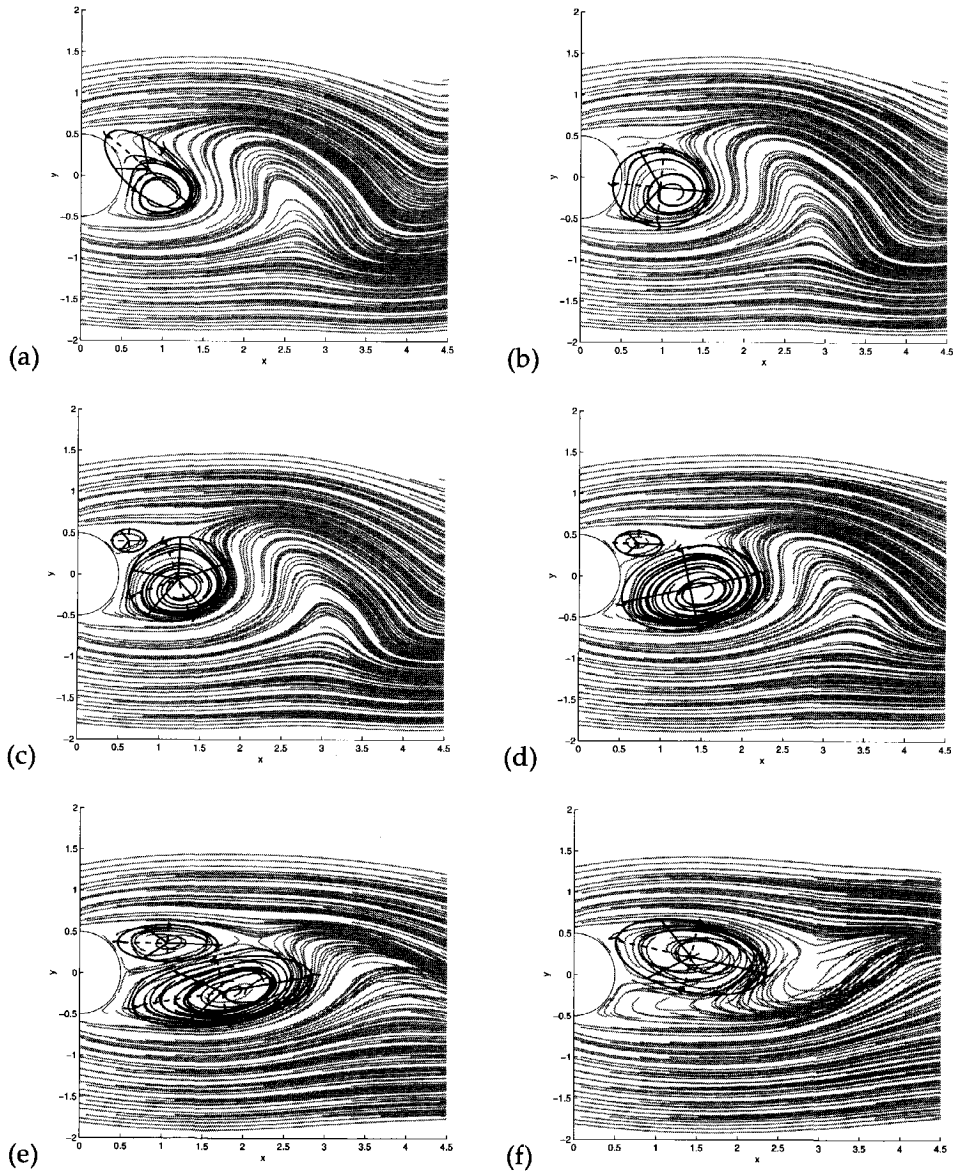


Figure 4.15: Flow past a tapered cylinder in different z-planes: $(a) : z = 16, (b) : z = 17, (c) : z = 18, (d) : z = 19, (e) : z = 20, (f) : z = 21$

which are not perfectly circular. Another important advantage is that it allows for quantification of vortices by calculating numerical attributes. The third major advantage of this method is that it also finds weak vortices, characterized by slow rotational movement. This will be shown in more detail in another application in Section 6.2.5.

4.6 Conclusions

We have shown examples of physical criteria and geometric criteria for vortex detection. Physical, point-based criteria are of limited use, because there does not seem to exist a generally accepted criterion, which works in all cases. An important drawback of physical criteria is that they usually do not detect weak vortices, because they often involve the velocity magnitude or another strength factor, directly or indirectly.

In principle, geometric, region-based criteria are better, because they correspond to the intuitive visual notion of a vortex, and because they are based on the properties of a whole region, such as a streamline or a number of streamlines. The curvature centre method performs well in finding (nearly) perfectly circular vortices, but it does not find elliptical and weak vortices. Possible enhancements to the curvature centre method have been investigated, but did not give a satisfactory result either.

The winding-angle method is much better, because it can detect not only non-circular but also weak vortices. Another important advantage is that it allows for quantification of vortices.

Chapter 6 will show more comprehensive examples of data sets to which these vortex detection criteria are applied.

Chapter 5

Deformable surfaces

In nova fert animus mutatas dicere formas corpora [1]

A deformable surface is a surface which grows from an initial shape to the shape of some feature in a data set, in order to detect that feature. The initial shape is a polygonal surface mesh, and growing is accomplished by displacing the nodes, based on the minimization of an energy or cost function for all nodes of the polygonal surface. When the polygons of the mesh become too large for an accurate approximation of the feature shape, the mesh may be refined. Then, deformation is continued, until some desired precision has been reached [Sadarjoen & Post, 1997; Sadarjoen & Post, 1999b].

Much other work has been done on deformable curves and surfaces, as was surveyed in Section 2.4. However, while traditional deformable curves and surfaces are usually intended for object detection in 2D or 3D images, our deformable surfaces have been specially intended for extracting features from 3D flow fields, in particular features which are locally continuous and can be well represented by surfaces.

This chapter first gives a definition of deformable surfaces and a description of their usage in Section 5.1. Section 5.2 describes how deformable surfaces are initialized. Then, Sections 5.3 and 5.4 discuss the mechanisms for surface deformation and surface refinement, respectively. Section 5.5 gives some example applications. Finally, in Section 5.6 a summary and conclusions are given.

5.1 Surface definition and usage

A deformable surface may be defined as a surface which starts from an initial shape and undergoes a deformation process to end in a final shape which approximates a feature in a data set. A deformable surface is typically defined as a polygon surface consisting of nodes, edges, and faces. The faces are usually triangles. By displacing the nodes, the surface is deformed.

A deformable surface may have a 1D, 2D, or 3D topology, where a 1D topology corresponds to a generalized cylindrical surface, a 2D topology to a planar surface, and a 3D topology to a spherical surface. Each triangular face has a normal, which in the 1D and 3D topologies point outward. At the nodes, normals are defined as the average of the normals of the adjacent faces.

[1] My soul drives me to speak of bodies who have changed into new shapes. (Ovidius, Metamorphoses I, 1-2)

The general procedure for surface deformation is as follows. A surface is locally deformed by displacing the nodes, using a node displacement criterion to govern the fitting of the surface to the data field. Our deformable surfaces are typically defined as small objects, which are made to grow by a deformation criterion. Whenever faces become too large, it may become necessary to *refine* them for a better shape approximation.

This procedure is illustrated in 2D in Figure 5.1. A contour line is shown, which is the feature to be extracted. Inside, a small initial deformable surface is positioned. After N deformation steps ($N \geq 1$), the surface is still too coarse, so it is refined, creating new faces and nodes. Then, deformation starts again, until a desired level of accuracy has been reached.

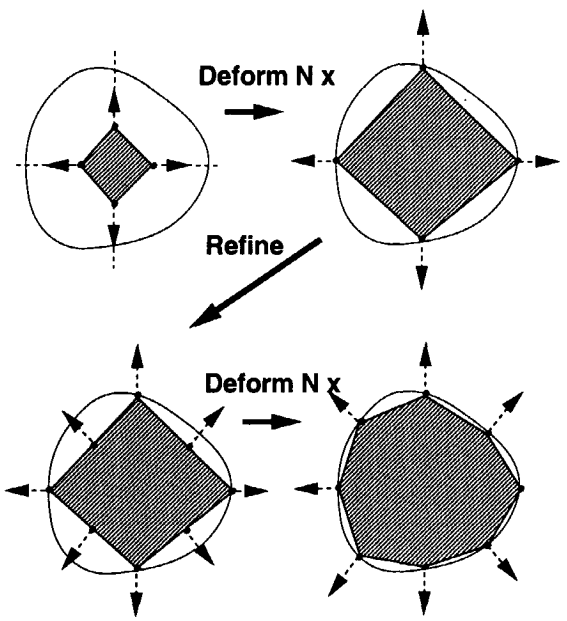


Figure 5.1: Overview of the deformation process. After N deformation steps, the surface is refined, and then deformed again.

Before we can apply the above procedure, some preparation steps are necessary. The complete process of using deformable surfaces consists of four stages: *region selection, initial surface creation, surface deformation,* and *surface refinement*. These stages can be represented by the pipeline shown in Figure 5.2.

In the first stage, a region is selected where the feature is assumed to be. In the next stage, an initial surface is created, based on the characteristics of the intended feature. In the third stage, the surface is actually deformed to approximate the intended feature from the data field. This may require several iterations, hence the loop back. In the fourth stage, the surface is refined. After this, it may be necessary to go back to the

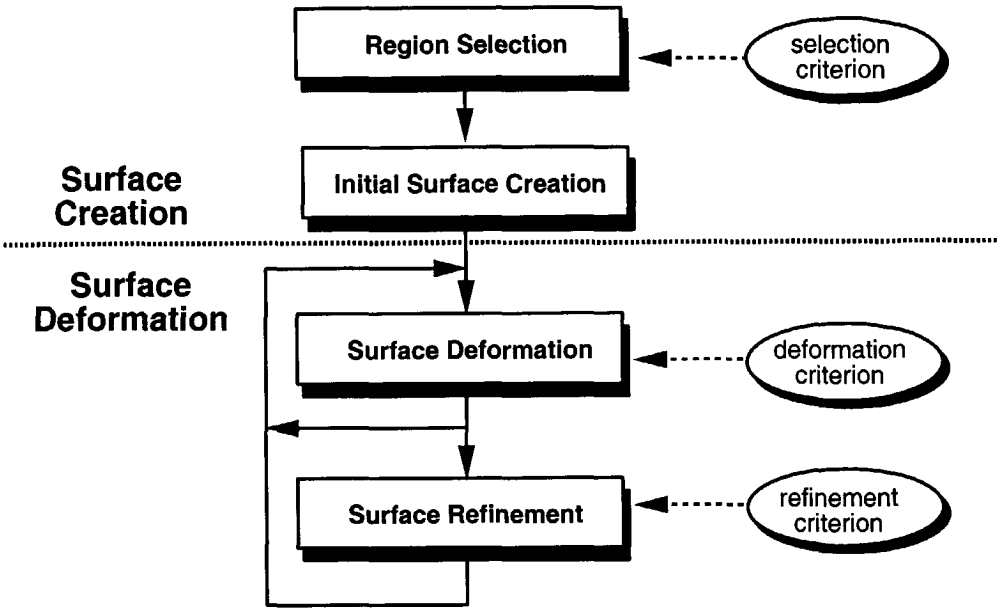


Figure 5.2: Overview of the complete process of using deformable surfaces

deformation when the desired accuracy has not been reached yet.

In three of these stages, the user can exert control by specifying application-specific criteria. In the first stage, a *region selection criterion* is used to determine the region where the centre of the initial surface should be located. In the surface deformation stage, a *deformation criterion* is used to steer the deformation process. In the surface refinement stage, refinement criteria and thresholds may be specified.

To implement this pipeline, we have developed a set of AVS-5 modules called GARIS. This acronym, which is also the Indonesian word for 'line', stands for Generic Adaptively Refinable Iterative Surfaces. The system contains modules for initial surface creation, surface deformation, surface refinement, and for interfacing with region selection modules. For the region selection, we have used existing modules developed by Van Walsum [van Walsum & Post, 1994; van Walsum *et al.*, 1996].

This pipeline is described in the following sections. Section 5.2 covers the initial surface creation stage, using region selection. Then, Sections 5.3 and 5.4 discuss the mechanisms for surface deformation and surface refinement.

5.2 Initial surface creation

Before surfaces may be deformed, first an *initial surface* must be created. An initial surface is determined by several *parameters*, the most important of which are shape

type, position, size, and orientation. The parameters of the initial surface must be related to those of the desired feature, as the final shape of the surface is determined through deformation of the initial surface.

Parameters may be determined manually or semi-automatically. In manual parameter determination, the user exploits experience and knowledge about the data and features in it, for determining the type, size, and position of the initial deformable surface. For this purpose, the GARIS system provides sphere, icosahedron, cube, plane, and cylinder primitives. The cube and icosahedron are rough approximations of a sphere; the other types should be self-explanatory.

The parameters may be determined semi-automatically through a method we have developed, where the user first manually specifies a selection criterion, after which the system automatically selects a region, calculates a fitted ellipsoid, derives the appropriate type (1D, 2D, or 3D) of deformable surface, and determines its position and scale parameters. This method is described next.

Starting from an input data set, a *region of interest* is selected using the selection techniques presented in [van Walsum & Post, 1994]. A *region selection criterion* is represented in a simple but powerful language, which allows users to formulate a selection criterion as a Boolean expression, and to select from the grid all nodes that satisfy the selection criterion. The criterion may use all raw data quantities, but also new quantities derived from the raw data, such as gradients. The Boolean expression may also contain logical combinations (and, or) or scalar thresholds on these quantities. The result is a selected subset of the grid nodes.

As an example, we select regions where the centripetal acceleration of a normalized velocity field exceeds a user specified threshold θ:

$$\mathbf{a}_c = \frac{\mathbf{v}}{\|\mathbf{v}\|} \cdot \nabla \left(\frac{\mathbf{v}}{\|\mathbf{v}\|} \right) > \theta$$

Figure 5.3 shows the expressions in the selection language to perform this selection. All expressions are evaluated at all nodes of the input grid. Each line except the last has the form *field* `a=` *arithmetic expression*, which calculates a new field variable using an arithmetic expression containing field variables. The last line has the form `select_out b=` *Boolean expression*, which creates a Boolean field that specifies for each node of the field whether it satisfies the Boolean expression. This Boolean field is generated as output data. The example also shows some built-in functions. The `len` function returns the length of a vector quantity, the `grad` function returns the gradient of a scalar or vector quantity, and the `mvm` function performs a matrix-vector multiplication. More information may be found in [van Walsum, 1995].

When a region has been selected, we calculate the center, variance, and covariance of the selected nodes, as these statistical attributes define an ellipsoid [van Walsum *et al.*, 1996]. The centre position (x, y, z), lengths (l_1, l_2, l_3) of the main ellipsoid axes, and the three rotation angles (α, β, γ) of the ellipsoid axes are used to characterize the shape of the selected region.

The shape of a region can now be classified as 1D, 2D, or 3D, where a 1D shape corresponds to a cylindrical shape, a 2D shape to a planar surface, and a 3D shape to a

```
normvelo a= velo/len(velo)
velograd a= grad(normvelo)
normaccel a= mvm(velograd, normvelo)
select_out b= normaccel > thresh
```

Figure 5.3: Selection of a region where $\mathbf{a}_c > \theta$

spherical volume. The shape of an ellipsoid is based on the ratios of the lengths of the axes l_1, l_2, and l_3. If the axis lengths are all in the same order, the region is classified as 3D, for which the ellipsoid itself may be directly used as the initial object. If one axis is small, and the other two axes are large, the ellipsoid is a flat, disc-shaped object. The region may then be classified as 2D, and the initial surface type chosen is a planar surface, fitted to the two long axes of the ellipsoid. If only one axis is large, and the two others are small, the ellipsoid is a long and thin object. The region is then classified as 1D, and the corresponding initial surface type is a (generalized) cylinder, with the axis fitted to the long axis of the ellipsoid, and an elliptic cross section of which the dimensions are based on the other axes.

This method of shape initialization has the limitation that it can only create objects based on straight lines, such as planar surfaces. To create good approximations of curved regions, such as banana-shaped regions, skeleton extraction algorithms could be used, but we have not implemented this. However, for our purposes, the ellipsoid fitting proved to be useful.

After the initial surface has been created, it is placed at the center of the selected region. The position, dimensions, and orientation of the initial surfaces are derived from the centre position, lengths, and orientations of the ellipsoid axes. Other parameters can be chosen by the user, such as the polygon mesh resolution, or colour mapping on the surface. Then, it can be deformed to its final shape. The deformation process is described in the next section.

5.3 Surface deformation

This section discusses the mechanisms and criteria for node displacement. Algorithm 1 shows the basic algorithm for surface deformation. Here, n is a node, $C(n)$ the cost function at a node n, $C(f)$ the cost function at the centre of a face f, f^* a marked face, and ε a threshold for the cost function C. $R(f)$ is a refinement criterion with a threshold θ.

Two important components of this algorithm are the *displace_nodes()* and *refine()* functions. The *refine()* function is covered extensively in Section 5.4. Algorithm 2 gives the pseudo-code for the *displace_nodes()* function, where $\mathbf{x}_n$ is the position of a node n, $\mathbf{x}_n^*$ a possible new position, $\mathbf{d}_i$ the step direction, and λ the step size. This function is described in detail in the remainder of this section.

Algorithm 1 Surface deformation

```
repeat
   repeat
      displace_nodes() (see Algorithm 2)
   until |C(n)| < ε for all nodes n.

   f* = ∅
   for all faces f do
      if R(f) > θ then
         f* = f*∪ mark_face (f)
         f* = f*∪ mark_neighbours (f)
      fi
   od
   for all marked faces f* do
      refine (f*)
   od
until |C(f)| < ε for all faces f
```

Algorithm 2 The *displace_nodes()* function

```
for all nodes n do
   if |C(n)| > ε then
      for i ∈ num_directions do
         d_i := determ_step_direction ()
         λ := determ_step_size ()
         x*_n := x_n + λ · d_i
         if C(x*_n) < C(x_n) then
            x_n := x*_n (move node)
         fi
      od
   fi
od
```

5.3.1 Node displacement

Node displacement is guided by a *deformation criterion*, which uses a *cost function* C based on the physical quantities given in the data set. These quantities may be scalars, vectors, or a combination. The criterion tries to minimize the value of C up to a specified tolerance ε.

Our cost function is comparable to the energy of the original snakes [Kass *et al.*, 1988], which is defined as $E_{tot} = E_{img} + E_{int} + E_{con}$ (see also Section 2.4.1). The total energy is composed of an image energy E_{img}, caused by features in the image, an internal energy E_{int}, giving the snake internal smoothness and stiffness, and a constraint energy E_{con}, imposed by the user, e.g. to make certain points fixed (see Section 2.4.1 for a more detailed desription). Both the snake energy and our cost function are to be minimized during the iteration process. However, our cost function does not contain an "internal energy" term. The reason is that we want our surfaces to be completely guided by the data field, and not by any autonomous smoothness constraint. For the same reason, our cost function does not incorporate a constraint energy term.

Our cost function is also comparable to the one used in the Geometrically Deformed Models (GDMs) [Miller *et al.*, 1991], which is defined as $C(x,y,z) = D(x,y,z) + I(x,y,z) + T_i$, where D represents a deformation potential, I an image term which identifies 'feature events', and T a topology-preserving term (see also Section 2.4.2). But our function does not contain an image term, because that presumes too detailed a-priori knowledge of the feature to be extracted. The T-term is comparable to the internal energy E_{int} of the snakes.

The simplest example of a cost function $C(\mathbf{x}_n) = Q(\mathbf{x}_n) - T$ evaluates the scalar field quantity Q at position $\mathbf{x}_n$ of a surface node, and determines the difference with a target value T. More complex cost functions use vector quantities, such as velocity or vorticity, which could e.g. be used to determine the angle between the local field vector and the local surface normal: $\angle(\mathbf{v}, \mathbf{n})$. If $\mathbf{v} \perp \mathbf{n}$ at all points of the surface, we have a stream surface.

Cost functions can be any function of quantities derived from the field values (e.g. gradients, vector operators, statistical attributes), and they may be combined with logical operators (and, or). Other examples of cost functions are given in Section 5.5 and Section 6.4.2.

However complex the deformation criteria and corresponding cost function may be, the result is always a *node displacement step*, which has two characteristics: a *step direction* $\mathbf{d}$ and a *step size* λ.

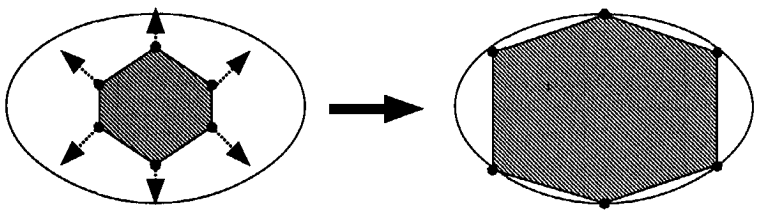
The *step direction* $\mathbf{d}$ may be determined in three ways:

1. perform a random search in several directions
2. follow the negative cost function gradient
3. follow the surface normal

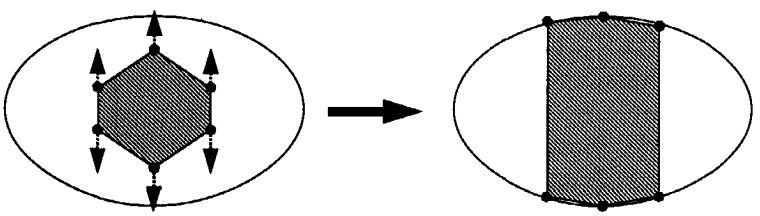
1. The random search method considers several candidate directions $\mathbf{d}_i$, e.g. in the direction of the principal coordinate axes and combinations thereof, and chooses the one that leads to the lowest cost function value. This method is often applied in 2D deformable contour applications, where all (4-connected or 8-connected) neighbours

of a pixel are examined, and the neighbour is chosen which has the lowest energy or cost function (see also Section 2.4.1). In 3D, this corresponds to 6, 18, or 26 directions which are tried.

Unfortunately, in the 3D case this method did not turn out to work well. The deformable surface has too much freedom, so the surface nodes individually choose the shortest path to the positions where the cost function $C(n)$ is minimal, which does not lead to uniform growth in all directions. This is aggravated by the fact that nodes are displaced over larger distances in 3D continuous space, rather than in 2D image space. This is illustrated by Figure 5.4, which shows an elliptic contour line in a scalar field and a deformable surface, projected 2D for simplicity. Rather than growing more or less uniformly in all directions, as shown in Figure 5.4a, the nodes on the deformable surface move along the shortest paths to the elliptic target contour, as shown in Figure 5.4b. which is not uniformly in all directions.



(a) desired behaviour



(b) actual behaviour

Figure 5.4: Determining the step direction by searching in many directions causes nodes to follow the shortest path, leading to undesired behaviour.

2. The cost function gradient method considers only one direction, the direction of the negative gradient of the cost function. This option was inspired by the so-called Steepest Descent methods well-known from optimization theory, which are efficient ways to find the minimum of a function defined on multidimensional domains [Press *et al.*, 1992], Sections 10.5 – 10.6). Unfortunately, this method did not turn out to work well with surfaces consisting of multiple nodes. The results are very similar to those

of the random-search method, although this method is faster, because only a single direction is examined rather than multiple directions.

3. The surface normal method also considers only one direction, the direction of the local surface normal, regardless of the local field gradient. This method turned out to work best, because it made the surface grow uniformly in all directions.

The *step size* λ may be determined in two ways: make the step size proportional to a local property, or perform a local search along a line.

The first way to determine the step size makes the step size proportional to some local property. A node is then displaced in multiple small steps, as illustrated in Figure 5.5. Hence, this method will be called the *multi-step* method.

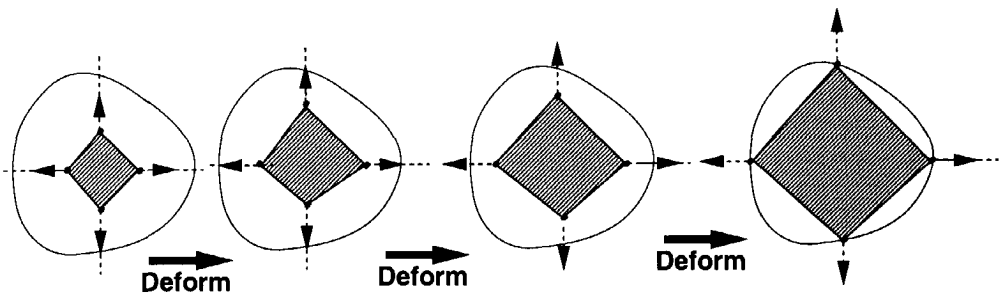


Figure 5.5: In the multi-step method, surface points are displaced by small steps at a time.

The local properties on which the step size can be based include:

1. grid cell size. This makes the step size proportional to the local grid cell size, usually as some fraction of the size of the local cell containing the node. This is applicable in curvilinear grids. The aim is to visit at least one or several points within each cell, in order not to miss important features in the data.
2. face size. This makes the step size proportional to the surface face size. The rationale is that at the end, when the faces have been refined several times, a small step size will prevent nodes from stepping too far (beyond the target surface).
3. cost function. This makes the step size proportional to the value of the cost function $C(\mathbf{x}_n)$. Initially, when C is highest, large steps are acceptable, but when the deformation progresses and C decreases, small steps will again prevent nodes from stepping too far.

The multi-step method suffers from two problems:

- slow iteration: typically, it requires many (tens to hundreds) of iterations for the surface to reach its final state. This is typical of many deformable contour methods.
- local minima: a surface sometimes does not (completely) reach its intended final state, because it gets caught in a local minimum. This is also a common problem for many other deformable contour or surface methods.

We have developed a fast and more robust alternative for the multi-step method, which is described in the next section.

5.3.2 Fast node displacement

The multi-step methods described above determine a large number of new positions $\mathbf{x}_n^* := \mathbf{x}_n + \lambda \cdot \mathbf{d}_i$, in search of the position where C is minimal.

Another, more efficient way to determine the step size uses a so-called *line search*: an efficient search along a line for the point where the cost function becomes zero:

$$C(\mathbf{x}_n + \lambda \mathbf{d}_i) = 0$$

and move the node directly to that position, as illustrated in Figure 5.6. Hence, this method will be called the *single-step* method.

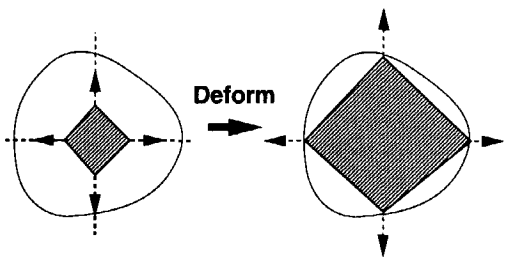


Figure 5.6: In the single-step method, surface points are placed on the target surface in one step.

The single-step method has two advantages: one is a considerable increase in speed of the iteration. Whereas deformable contour methods typically require tens or hundreds of steps to reach their desired shape, the single-step method requires only several steps. This will be illustrated later in the examples in Section 5.5. Another advantage is that the problems of local minima are usually solved.

To find the root of the cost function C, we use the Regula Falsi, also known as the False Method [Press *et al.*, 1992]. This method finds a root of a 1D function by successive interpolation between two 'brackets' where the function values have opposite signs, as illustrated in Figure 5.7.

While the standard literature assumes that the brackets are known in advance, this is not the case when using deformable surfaces. For the left bracket λ_l, the current position of the surface node can be taken. However, the right bracket still has to be determined. Our solution is to estimate the right bracket λ_r by performing one step with the Newton-Raphson method. Essentially, this extrapolates the tangent in λ_l. If the interval given by the left and right brackets does not contain the root, it may be extended as often as necessary. Once the brackets have been determined, the root is estimated by interpolation (points λ_1, λ_2, and λ_3 in Figure 5.7), and the interval is successively reduced until the root has been approximated with the desired accuracy.

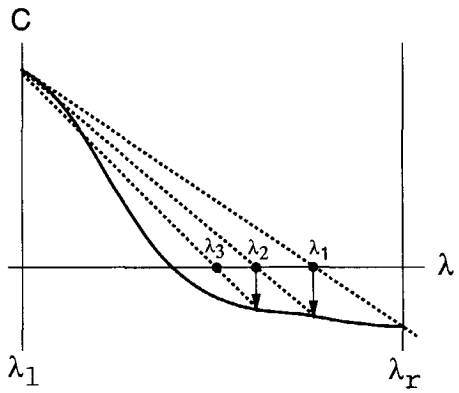


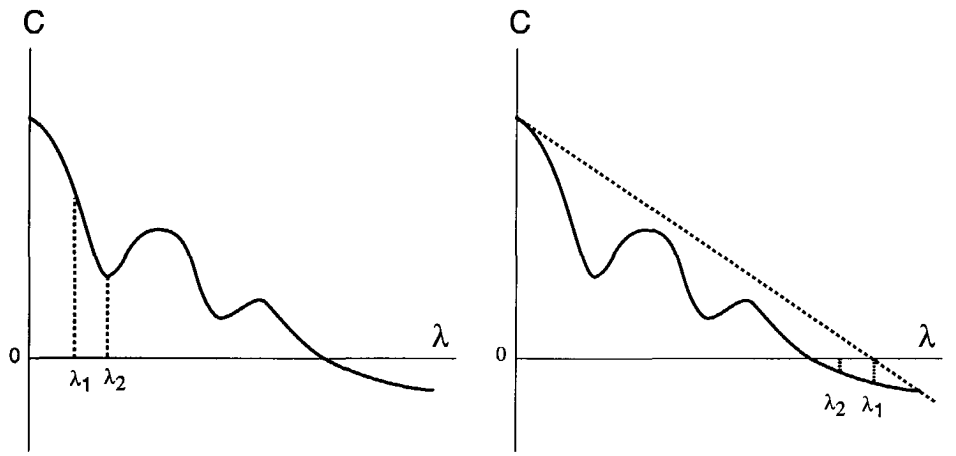
Figure 5.7: Root-finding using Regula Falsi between brackets λ_l and λ_r yields successive approximations of the root: $\lambda_1, \lambda_2, \lambda_3, \ldots$

Both the multi-step and the single-step methods have their advantages and disadvantages. The multi-step method has the disadvantage that it is slower and more sensitive to local minima. Figure 5.8a shows how a node is moved over small consecutive distances λ_1 and λ_2, but then the node gets stuck in a local minimum of the cost function $C(\lambda_2)$. From that point, all the points in a small neighbourhood $(\lambda_2 - \Delta\lambda, \lambda_2 + \Delta\lambda)$ keep the node in the local minimum.

In contrast, the single-step method has the advantages that it is much faster and less sensitive to local minima. Figure 5.8b shows that the local minima are skipped, when an initial guess is taken for the right bracket of the Regula Falsi method (the extrapolation from C(0)). However, the line-search has the disadvantage that it gives nodes a much greater freedom to move over larger distances in one step. This freedom, combined with the complete independence of adjacent nodes, may lead to erratic behaviour of nodes, and unpredictable surface shapes, when nodes are moved with (very) different step sizes. In addition, the estimation of the right bracket with the Newton-Raphson method is sensitive in regions of high gradients (field derivatives), because it depends on a single gradient direction at λ_l.

5.4 Surface refinement

When the faces of a deformable polygonal surface become too large, such that the surface no longer approximates the feature very well, the surface can be refined, so as to improve the approximation of the feature to be extracted. This section discusses criteria and mechanisms for refinement.



(a) multi-step method (b) single-step method

Figure 5.8: The multi-step method gets caught in a local minimum λ_2 of the cost function, while the single-step method jumps over local minima.

5.4.1 Refinement criteria

First of all, we can distinguish between global and local refinement. In global refinement, all faces of the surface are refined. This is straightforward, but costly both in terms of efficiency and storage requirements. In local, or adaptive refinement, only those faces f are refined for which a *refinement criterion* $R(f)$, based on quantities discussed below, exceeds a specified threshold θ: $R(f) > \theta$. This threshold θ is a relative threshold (between 0 and 1) specified by the user.
We have investigated refinement criteria based upon the following quantities:

1. face size
2. local curvature
3. cost function $C(f)$
4. estimated distance

A refinement criterion based on the first quantity marks the largest faces of the deformable surface. The motivation is that it is undesirable to have large faces, because they may fail to follow the curvature of the target feature.

Therefore, a refinement criterion based on the second quantity marks faces where the local curvature is highest. However, the goal is not to minimize the curvature over the entire surface, as in other applications. The goal here is to obtain the best approximation of a feature in the field. Therefore, the cost function is a better base for the refinement criterion.

A refinement criterion based on the third quantity marks those faces that have a

high cost function $C(f)$ at their centres. The scalar cost function for a face is evaluated at the face *midpoint* as the average of the cost function values at the nodes. The rationale is that these faces are far from the target surface. This turned out not to work completely as expected: sometimes there are faces whose cost function is large, but which are physically close to the target surface. This occurs when there are locally high gradients.

The last refinement criterion marks those faces that have a high distance to the target surface, which is estimated by

$$\frac{C(f)}{\|\nabla f\|}$$

This criterion gives the best results, in the sense that the best shape approximation of features was obtained. This success could be explained from the fact that it uses the best measure for deviation of the deformable surface from the target surface.

5.4.2 Refinement mechanisms

There are several mechanisms for surface refinement. We assume that the surface is represented as a triangle mesh, and that the refinement criterion has marked some of the triangles for refinement.

One class of mechanisms to perform mesh refinement, which we have implemented, is described in [Rivara, 1991] and [Mitchell, 1991]. Both authors describe methods which rely on bisection of a triangle at its longest edge, where a new node is added. This may lead to a so-called *hanging node* in an adjacent triangle, which then becomes *inconsistent*: one of its edges has 3 nodes instead of 2. In [Rivara, 1991], this is solved by applying the bisection to the adjacent triangle, and repeat this as long as necessary until the new node coincides with the first hanging node. This is done using a recursive procedure. Figure 5.9 illustrates how first the marked face ABC is bisected, along the (dashed) longest edge. This creates a new midpoint node P. As face ABD is now inconsistent, this triangle is then bisected along longest edge AD, which creates new node Q. Finally, ABQ is bisected, with the new midpoint node coinciding with node P.

In contrast, the method described in [Mitchell, 1991] tries to prevent hanging nodes from being created at all, by refining only *pairs* of triangles which are so-called *compatibly divisible*. This means that they either share a longest edge, or one of them is part of the boundary. In the first case, the two triangles can be refined as a pair; in the second case, the two triangles can be refined after a single bisection of the boundary triangle.

See Figure 5.10 for an example, where the marked face ACE has to be refined. As its longest edge CE is not the longest edge of its neighbour triangle CDE, the two triangles are not compatibly divisible. Since CDE is part of the boundary, after a single bisection into CDG and CEG, the new triangle CEG now shares its longest edge with ACE, and the two can be refined as a pair.

This method claims to be more efficient than Rivara's. However, this method also uses a recursive procedure to find two compatibly divisible triangles first.

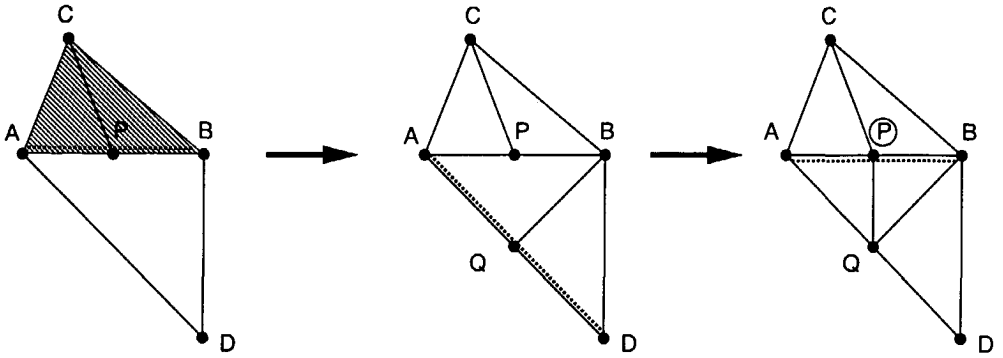


Figure 5.9: Rivara's method: first bisects the marked face ABC, then face ABD, and finally face ABQ. The longest edges are dashed.

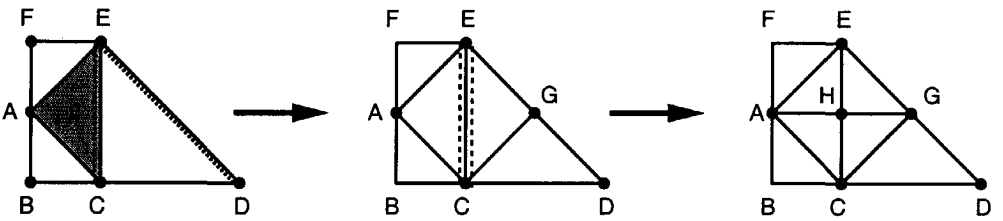


Figure 5.10: Mitchell's method only refines pairs of triangles that are 'compatibly divisible', to prevent inconsistent triangles from occurring.

When we implemented these schemes in GARIS, the problem of cycles occurred. In recursively marking the adjacent triangles, the algorithm encountered the same triangle where it started, which resulted in an infinite recursion. This was probably caused by equilateral triangles; in accordance with the original algorithm, the longest edge was chosen as the refinement edge, which is of course ambiguous in equilateral triangles. A solution to this ambiguity problem does not seem trivial.

Another refinement mechanism we have implemented is the one described by [Miller, 1990], and illustrated in Figure 5.11. A marked face (hatched in the figure) is refined into four triangles, and the adjacent faces, where hanging nodes would occur, are bisected into two triangles.

Care must be taken when faces are adjacent to more than one marked face, as shown in Figure 5.12. Here, the two faces would cause different bisection directions in their respective adjacent faces. This is solved by not bisecting such a face, but refining it into four faces as well, as if it had also been marked.

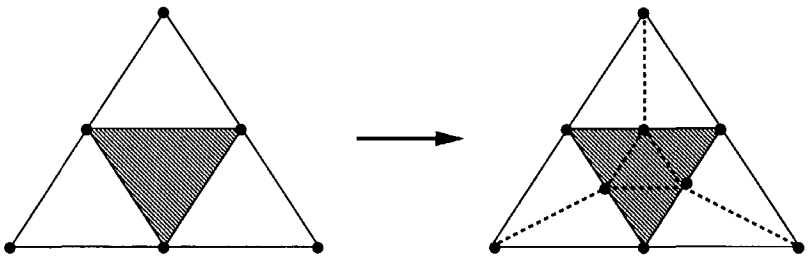


Figure 5.11: Miller's method refines the marked face into four faces and bisects the adjacent faces.

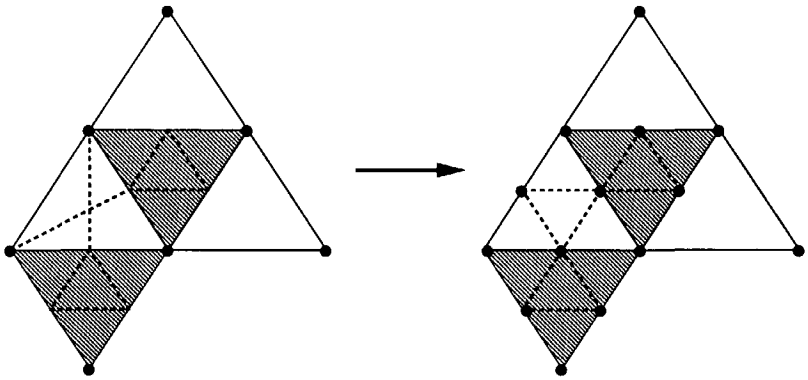


Figure 5.12: A face adjacent to multiple marked faces is not bisected, but refined into four triangles.

5.5 Examples

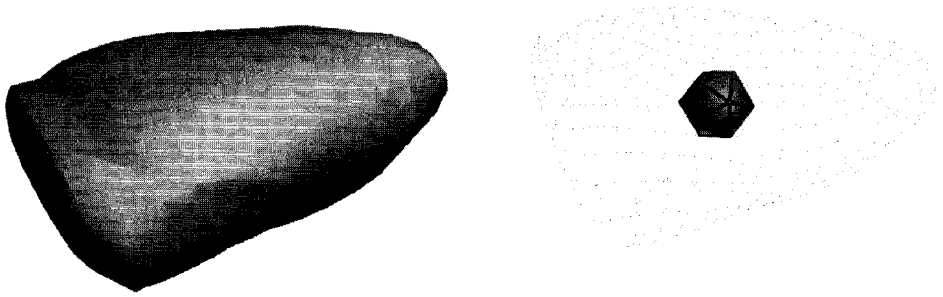
Next, we show two example applications of deformable surfaces. In Section 5.5.1, we show a local isosurface approximation which uses fewer triangles than a standard isosurface algorithm. Then, in Section 5.5.2, we show an application with more complex criteria, the extraction of a recirculation zone.

5.5.1 Extracting local isosurfaces

To illustrate the usage of the technique, deformable surfaces are used to approximate local isosurfaces in a 3D Backward-Facing Step (BFS) (see also Section 3.5.1).

Figure 5.13a shows a local isosurface of an arbitrary scalar quantity in this data set, in this case the x-component of the velocity vector. This isosurface was generated by the standard `isosurface` AVS-5 module to serve as a reference. Figure 5.13b shows a deformable surface in its initial shape, in this case an icosahedron as a rough approximation of a sphere. It has been manually positioned near a local maximum of the

scalar field. In this figure, the reference isosurface has been rendered as dots.



(a) reference isosurface (b) initial deformable surface

Figure 5.13: Approximation of an isosurface by a deformable surface in a 3D Backward-Facing Step

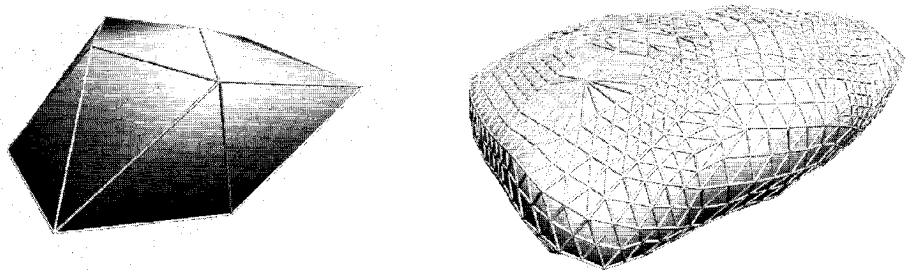
This deformable surface is now iterated with a simple displacement criterion, where the cost function is the difference between the local scalar field value and the target isolevel: $C(\mathbf{x}) = f(\mathbf{x}) - T$. The displacement step direction is taken in the direction of the surface normal. The step size is determined in two ways: with the multi-step method and with the more efficient single-step line search method. Refinement is done with a criterion based on the estimated distance to the target surface, as described in Section 5.4: $R(f) = d > 0.6 \max(d)$.

Figure 5.14a shows the shape of the deformable surface after 17 iteration steps with the multi-step method. This is the number of deformation steps before the first refinement step (in this particular case). Figure 5.14b shows the deformable surface after 48 iteration steps and 4 refinement steps. The surface now consists of 2520 faces. In order to show the surface refinement, the face edges have also been drawn.

In contrast, Figure 5.15a shows the shape of the deformable surface after one iteration step with the single-step method. The surface nodes already lie on the reference surface, and are ready to be refined. Figure 5.15b shows the final shape of the surface after only 4 iterations and 4 refinement steps. The surface now consists of 1720 faces.

Figure 5.16 shows the final deformable surfaces without the edges rendered, to allow for the best comparison with the reference isosurface. It can be seen that the surfaces approximate the reference surface of Figure 5.13a quite well; the single-step method even slightly better than the multi-step method, even though the latter consists of more faces.

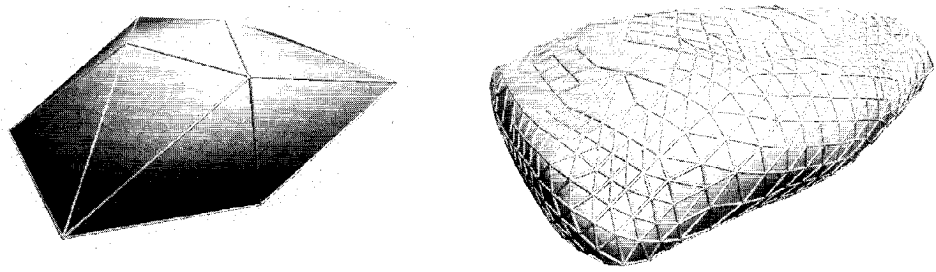
An advantage over regular isosurfaces is that the deformable surfaces consist of fewer triangles: 1720 and 2520 for the multi-step and single-step deformable surfaces,



(a) shape after 17 iterations

(b) shape after 48 iterations

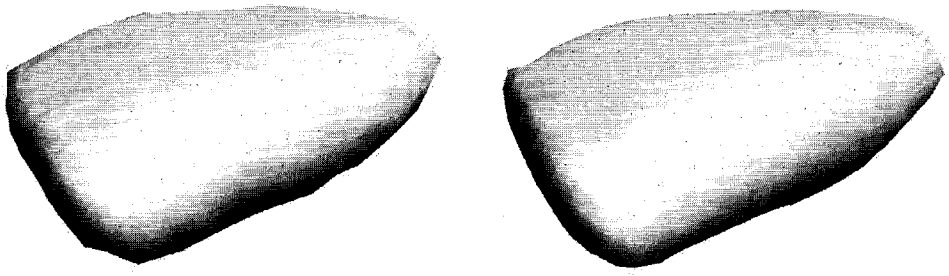
Figure 5.14: Approximation of the isosurface using the multi-step method



(a) shape after 1 iteration

(b) shape after 4 iterations

Figure 5.15: Approximation of the isosurface using the single-step method



(a) final shape with multi-step method

(b) final shape with single-step method

Figure 5.16: Approximation of the isosurface using the multi-step method

respectively, versus 4450 for the regular isosurface.

5.5.2 Extracting recirculation zones

In the 3D Backward-Facing step, we now try to extract the recirculation zone, as shown in Figure 3.15 in Section 3.5.1.

Our method of extracting recirculation zones consists of two steps: the first step uses a selection criterion to find the centre of the zone and to position the initial deformable surface. The second step uses a deformation criterion to find the separating surface between the recirculation zone and the main stream zone.
Following the scheme in Figure 5.2, first region selection and surface creation are performed to create an initial deformable surface. The criterion used to select a region of interest is $H_n > 0.6$, with H_n being the normalized helicity as defined in Section 4.1.

$$H_n = \frac{\mathbf{v} \cdot \boldsymbol{\omega}}{\|\mathbf{v}\| \cdot \|\boldsymbol{\omega}\|}$$

Figure 5.17 shows how this can be expressed in the selection language. The structure is similar to that of Figure 5.3. In addition to the `select_out b=` function, there is a `field_out a=` function, which sends the computed quantity to an output port of the AVS module. The `rot()` function returns the rotation (curl) of a vector quantity, the `dot()` function returns the dot product of two vectors, and the `fabs()` function returns the absolute value of a floating point quantity.

Through the selected regions, an ellipsoid is fitted, of which the statistical attributes mean, variance, and covariance define an ellipsoid. Figure 5.18 shows a combined view of the selected nodes rendered as cross marks, and the fitted ellipsoid. From

```
velorot a= rot(velo)
velohd a= dot(velo,velorot)
velohdn a= velohd/(len(velo)*len(velorot))
select_out b= fabs(velohdn) > 0.6
field_out a= fabs(velohdn)
```

Figure 5.17: Calculation of normalized helicity

the ratios of the ellipsoid axes, an initial deformable surface is derived, which is the cylinder shown in Figure 5.19.

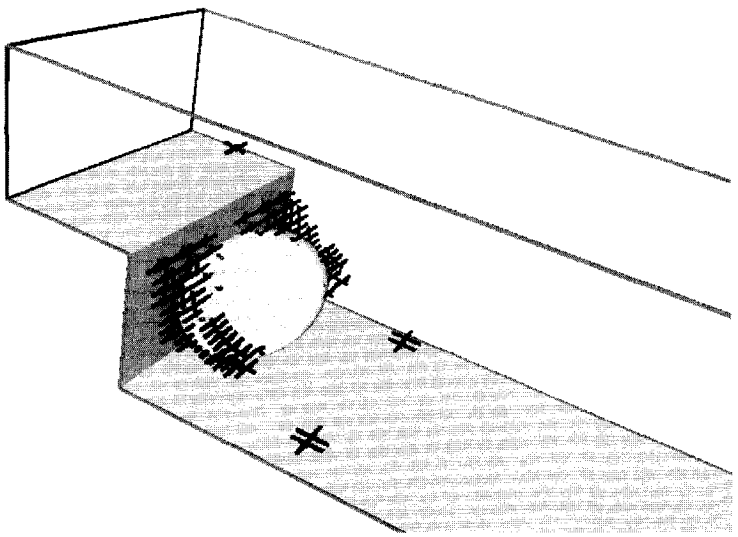


Figure 5.18: Selected nodes and fitted ellipsoid

See also Colour Plate 3, which shows a combined view of Figures 5.18 and 5.19, with the selected nodes and the deformable surface. The deformable surface is coloured with the cost function $C(\mathbf{x}_n)$ normalized between 0.0 and 1.0, with red indicating the highest, and blue the lowest value.

This initial surface is deformed using a criterion with a cost function based on a high gradient of the velocity magnitude, $g(\mathbf{x}) = \|\nabla(\|\mathbf{v}(\mathbf{x})\|)\|$:

$$C(\mathbf{x}) = \max(g) - g(\mathbf{x})$$

This cost function proved to be suitable for detecting the recirculation zone with a deformable surface, because it exhibits a 'valley' in the region separating the recirculation zone and the main stream. So, by minimizing this cost function, the initial surface deforms to a separating surface.

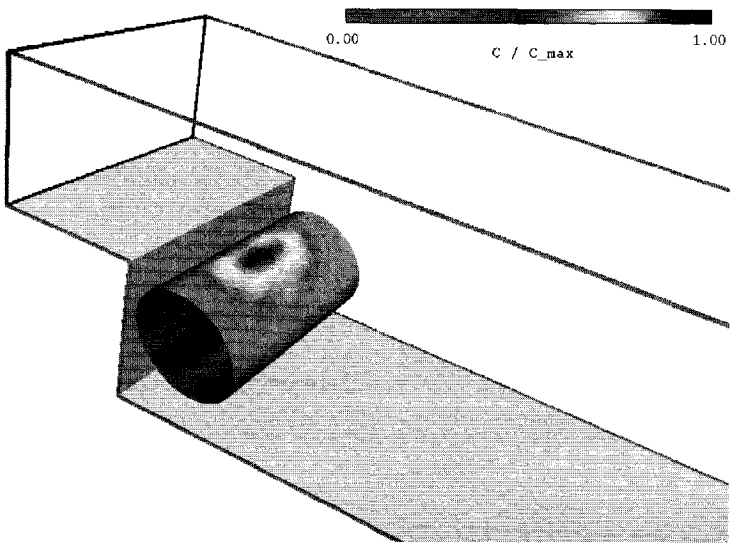


Figure 5.19: Initial deformable surface

Figure 5.20 shows the final surface of the recirculation zone after one iteration with the single-step iteration method. No subsequent refinement steps were necessary, as the initial resolution was chosen high enough.

The horse shoe shape of the lowest part of the surface is comparable to the separation / reattachment lines seen in other analyses of the BFS, such as the 0.0 contour on the floor (behind the step) in Figure 5.21. This figure shows contours of the streamwise velocity component, where the 0.0 contour is a separation line between flow going downstream, with a positive streamwise velocity, and the reversed flow going into the recirculation zone, with a negative streamwise velocity component (see also Figure 3.15).

Colour Plate 4 shows the same figure in colour, where the deformable surface has been coloured with the cost function. The uniform blue colour (cost function $C(f) \approx 0$. clearly shows that the upper part of the surface approximates the target surface quite well. The lower part of the surface has expanded as far as it could, but the grid boundary prevents the surface from expanding any further, so it is not blue. The surface consists of 240 triangles and takes 3.2 s to generate on an SGI Indy workstation, which makes it suitable for use as an interactive tool.

Another more comprehensive example of the use of deformable surfaces will be shown in Section 6.4.

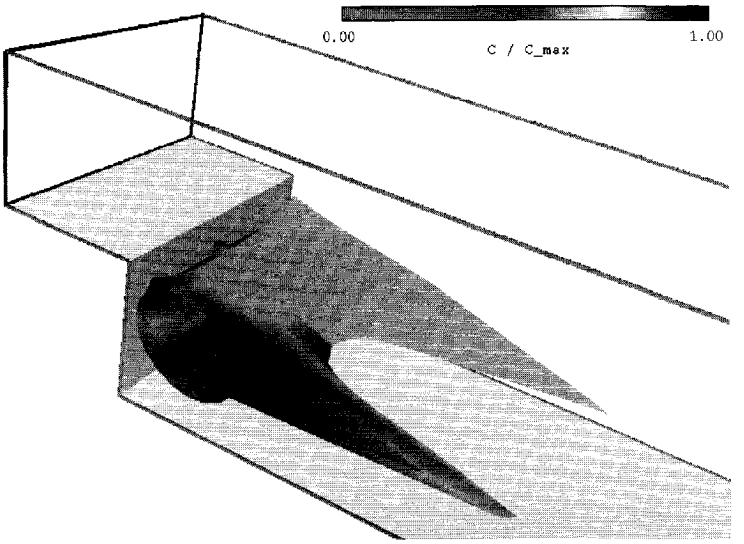


Figure 5.20: Final deformable surface approximating the recirculation zone

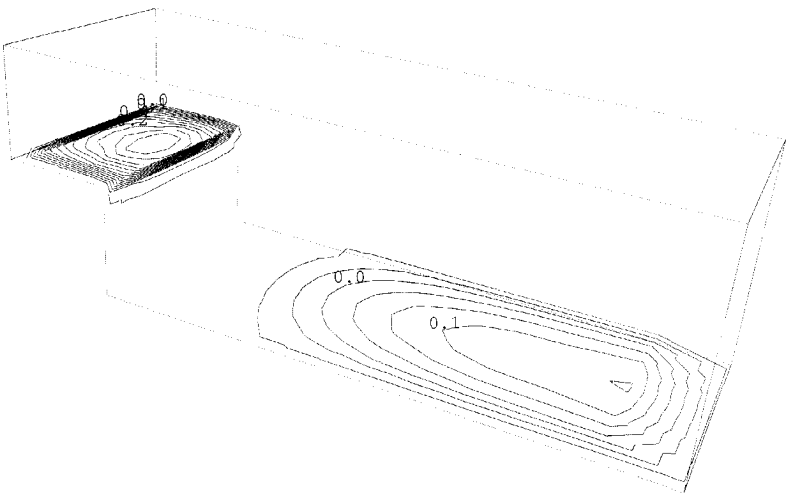


Figure 5.21: Contours of streamwise (y-)velocity

5.6 Conclusions

In this chapter, we have shown techniques and applications of deformable surfaces. We have described mechanisms for surface deformation and surface mesh refinement. For surface deformation, two schemes have been investigated. One is based on traditional methods applied in deformable contour / surface techniques. In addition, we developed an improvement which works by taking much larger steps. For surface refinement, we have investigated several schemes. Two schemes are based on recursive bisection of triangles. Another scheme is based on a single-step refinement into four or two triangles. Finally, we have shown examples of the use of deformable surfaces for extracting local isosurfaces and recirculation zones in a 3D backward-facing step.

Chapter 6

Applications

大江來從萬山中
山勢盡與江流東
鍾山如龍獨西上
欲破巨浪乘長風[1]

This chapter presents four applications, in which actual CFD simulations are analysed and visualized using the techniques described in the previous chapters. This is done with two goals in mind. One goal is to illustrate how the techniques can be applied and how they perform. When several alternative techniques are available for the same purpose, we will make a comparison. Another goal is to show which insight in the underlying physical phenomena can be gained by applying the visualization techniques.

The techniques from the previous chapters: particle tracing, vortex detection, and deformable surfaces may be applied with the following purposes. Particle tracing may be used to get an overview of the flow patterns in the flow field, by releasing particles in all the nodes of a grid slice. Then, vortex detection techniques may be used to find vortices in a more directed manner. Finally, deformable surfaces may be applied for detecting surfaces, in particular separating surfaces.

These techniques are applied to cases resulting from flow simulations of:

- the Pacific Ocean
- the Bay of Gdańsk
- turbulence in a cylindrical pipe
- a Delta-Wing aircraft

The following sections in this chapter describe for each case: the background, the data set, the applied techniques, the results, and a summary.

6.1 The Pacific Ocean

The first application concerns a numerical simulation of the Pacific Ocean [Zhu & Moorhead, 1995]. The simulation employs the US Navy layered ocean model, a tool used by the Naval Oceanographic and Atmospheric Research Laboratory to assist in ocean prediction. Horizontally, the model covers the area from 110°E to 78°W. Vertically, the model consists of six layers. The simulation models a period of four years,

[1]The Great River approaches from 10,000 mountains // The power of the mountains flows east with the River // The Clock Mountain seems like a dragon turning west // Backed by the storm wind, it breaks the high waves. (Gao Qi)

with a resolution of 3.05 days between time steps. The goal was to investigate the migration and interaction of ocean eddies over time and space, which is important for oceanographers to understand ocean circulations. The grid used is a rectilinear 2D grid of 468 × 336 nodes. At each node, a 2D velocity vector is given, and a Boolean indicating whether the node is located in the flow area or a land point, where the velocity is zero. This allows arbitrary coast geometries to be modelled.

We use one time step of this data set, and a subregion covering the west coast of North-America, ranging from 170°W to 110°W and from 35°N to 62°N, which includes Alaska in the northwest, and the Lower California peninsula in the southeast. This corresponds to an area of approximately 6500 km ×3000 km. Of the 3D grid, we use the surface layer in a 1:4 scaled-down version with 117 × 84 nodes.

6.1.1 Streamlines

To visualize the global stream patterns, we use streamlines. Since the grid in this data set is rectilinear, streamline calculation is straightforward, and does not require any special algorithms such as the ones discussed in Chapter 3.

Streamlines are released from every other grid point. The step size and the number of the steps for the integration of the streamlines must be chosen carefully. The step size should be chosen as small as possible to achieve the highest accuracy, and the number of steps should be as high as possible to ensure that paths are long enough, particularly in regions of low velocity magnitude. Here, we use 127 time steps of constant size $\Delta t = 2s$; these optimum values were determined empirically.

Figure 6.1 shows the resulting visualization of the global flow patterns using streamlines. It can be seen that there are many vortices.

6.1.2 Vortex detection with scalar criteria

To see how well scalar quantities indicate the presence of vortices, we use some of the quantities mentioned in Section 4.1. From the velocity field, we derive the vorticity ω, and λ_2, the second-largest eigenvalue of $\nabla \mathbf{v}$. As the data sets are 2D, we use the 2D versions of these quantities: in 2D, ω is a scalar quantity, and the velocity gradient $\nabla \mathbf{v}$ has only two eigenvalues, but the $\lambda_2 < 0$ criterion can be used in the same way as in 3D.

Figure 6.2 shows the vorticity and λ_2 scalar fields as height fields. To make the peaks easier to discern, each point has been coloured with its height. Note that for the λ_2 field, we visualize $-\lambda_2$, since we are interested in regions where λ_2 has a large negative value. As these regions would become valleys in the height field, which are harder to see than peaks, we use the negated quantity.

It can be seen that the vorticity field fails to make a sufficiently clear distinction between the vortices and the "background", because it contains too many (false) peaks. Apparently, high vorticity is not only caused by vortices. The λ_2 field does make a clear distinction between the vortices and the background, but it contains too few peaks. Apparently, not all vortices "create enough λ_2". This case shows that these

Figure 6.1: Pacific Ocean; streamlines

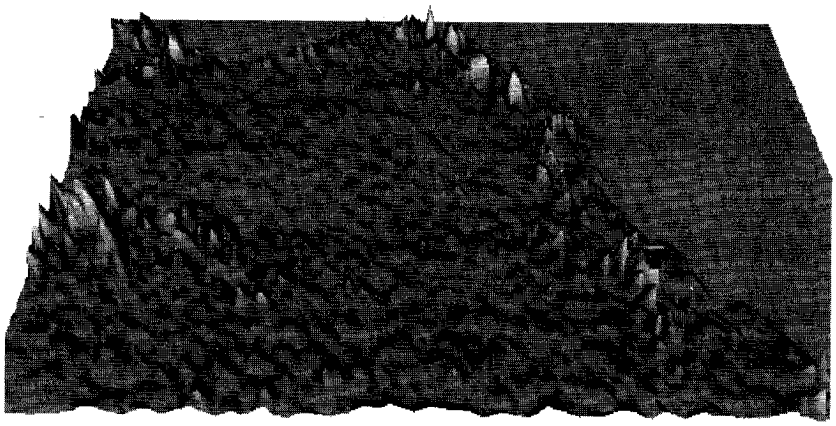
scalar quantities are not good indicators of vortices, because the quantities are point-based (see Section 4.1), while vortices are regional phenomena.

6.1.3 Vortex detection with critical points

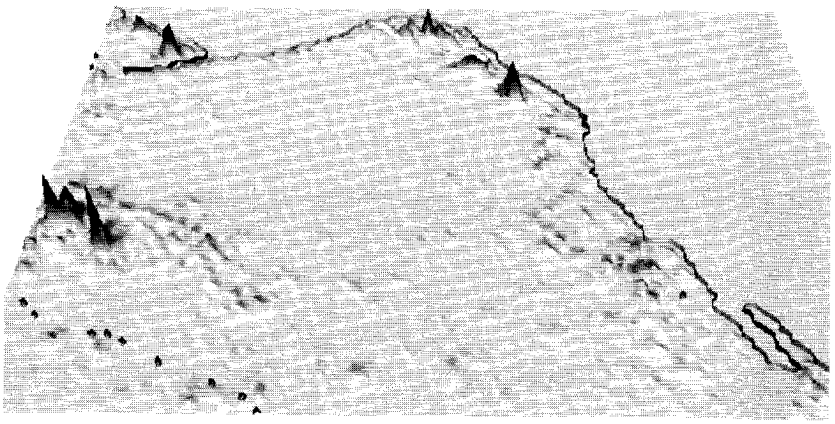
We have also calculated critical points for finding vortex centres. Figure 6.3 shows the critical points in the same data set, with different symbols for the various types. Unfortunately, the figure shows many more critical points than obvious vortices, including many false positives. So, unfortunately, there does not seem to be a one-to-one correspondence.

6.1.4 Vortex detection with curvature centres

The results of applying the curvature centre method and the enhanced curvature method were shown in Sections 4.3 and 4.4, and will not be discussed any further in this chapter.



(a) Vorticity



(b) λ_2

Figure 6.2: Pacific Ocean; scalar criteria for vortex detection

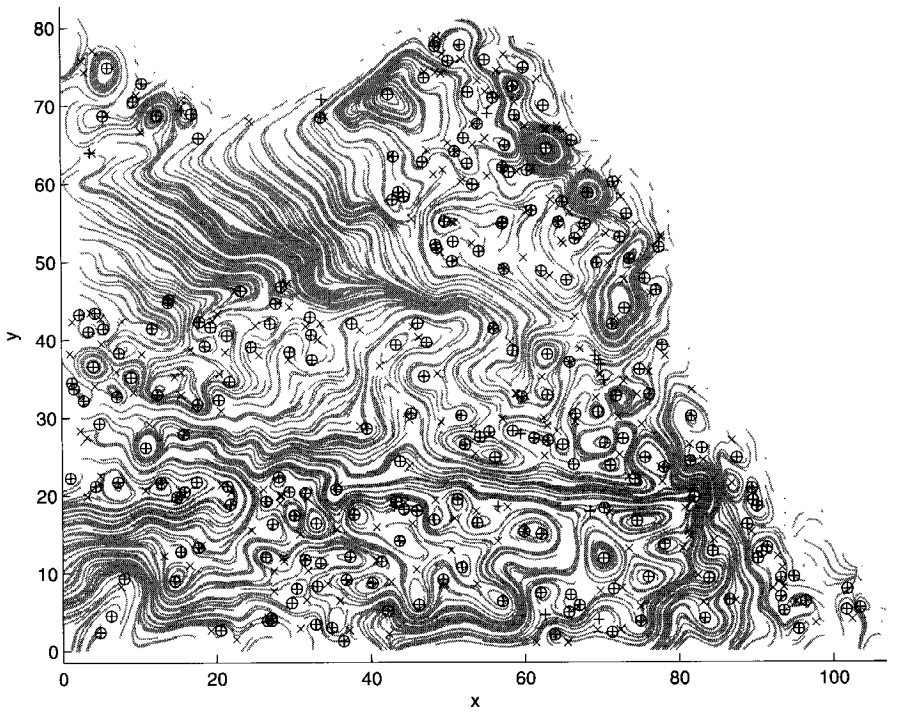


Figure 6.3: Pacific Ocean; critical points indicating vortex cores. 'o' are centres, '⊕' are foci, '+' are nodes, and '×' are saddle points.

6.1.5 Vortex detection with the winding-angle method

As an alternative to the curvature centre method, we apply the winding-angle method described in Section 4.5 to the streamlines in Figure 6.1. The selection criteria were: minimum winding-angle $\alpha_w = 1.5\pi$, maximum absolute distance $d_{max,abs} = 5$ grid nodes, and maximum relative distance $d_{max,rel} = 3 \cdot r_{est}$, where r_{est} is the estimated radius (see Section 4.5.1).

Once streamlines have been selected, they are clustered (see Section 4.5. Streamlines of the same cluster are considered to belong to the same vortex. In this case, we use a cluster radius of 3 grid cells. Clustering the selected streamlines also allows for quantification of the vortices, by calculating numerical attributes, some of which are listed in Table 6.1.

Figure 6.4 shows how ellipses can visualize the numerical attributes of the vortices. The ellipses show the approximate size and shape of the vortices, and the ellipse axes,

Number of clusters	64
Number of CW vortices	29
Number of CCW vortices	35
Min. radius [grid cells]	0.044105
Max. radius [grid cells]	4.604711
Min. $\omega[s^{-1}]$	0.037644
Max. $\omega[s^{-1}]$	0.271305

Table 6.1: Pacific Ocean; numerical vortex attributes

drawn in dashed lines, show the orientation of the vortices. The number of spokes indicates the strength of the vortices: the higher the number of spokes, the faster the rotation. Arrow heads show the rotation direction.

It is clear that this method produces the best results so far, since it detects both slow vortices and elongated vortices, which remain undetected by the scalar criteria and by the curvature centre methods.

A minor drawback is that this method does not detect extremely weak vortices, which have very short pathlines. There is a minimum length for the pathlines before they can be detected by this method. Usually, this is not a problem, and all the vortices are detected. But when too many weak vortices remain undetected, the pathline lengths can be increased, by increasing the number of integration time steps (or the step size, but this usually decreases the accuracy).

This application has shown results of vortex detection with physical criteria and with the winding-angle method. The physical criteria turned out to be moderately useful; the winding-angle was much better and allowed for quantification of the vortices.

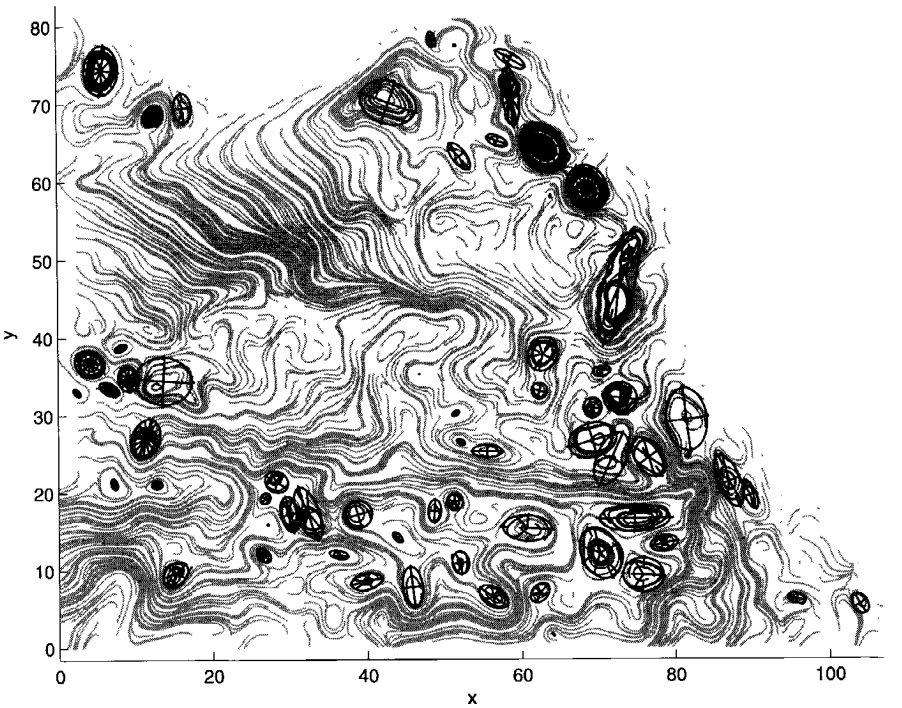


Figure 6.4: Pacific Ocean; vortices visualized by ellipses

6.2 The Bay of Gdańsk

This application concerns a simulation performed at WL | Delft Hydraulics of the Bay of Gdańsk, a coastal area of 192 × 107 km in northern Poland, using TRISULA their in-house developed software [Trisula User Guide, 1993]. This simulation was studied earlier by Hin for visualization of turbulence using particles [Hin & Post, 1993; Hin, 1994]. The flow in this model is driven by wind and an inflow of the Wisła (Vistula) river which enters the bay from the south.

The model is defined on a curvilinear grid of 43 × 28 × 20 nodes, which is in fact a σ-transformed grid consisting of 20 layers of 43 × 28 nodes. For a description of σ-transformed grids, see Section 3.3.1. Figure 6.5 shows two views of the grid: (a) a top view of a horizontal grid slice, and (b) a side view with the sea bed and a vertical grid slice. At each grid node, the simulation has computed a velocity vector $\mathbf{v}$, an eddy diffusivity scalar E, which represents turbulence intensity, and its gradient ∇E.

6.2.1 Particle tracing

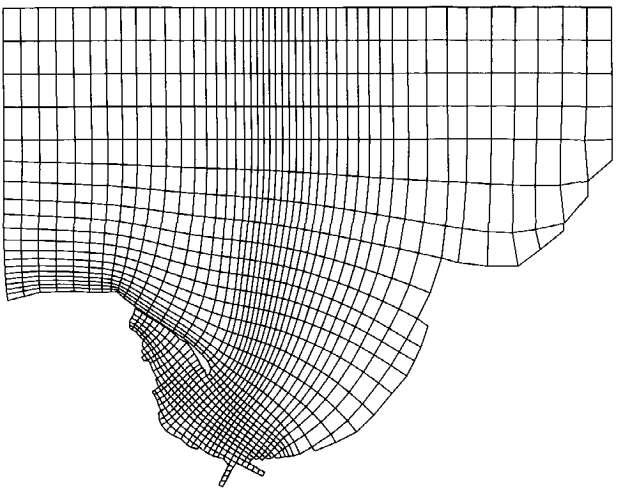
We apply particle tracing to visualize the global flow patterns in a horizontal 2D slice at the centre of the grid ($k = 9$). To obtain a good impression of the global flow patterns, particle sources are located at every grid node, and particles are traced for up to 100 time steps of $\Delta t = 400s$.

Figure 6.6 shows the resulting particle paths in white, to focus on the shape of the patterns: a number of distinct vortical regions are clearly visible, and some of them are very non-circular. Colour Plate 5 shows the same particle paths coloured with the velocity magnitude. Red indicates high values, and blue low values. The figure shows that some vortices are much slower than others, something which can also be seen in particle animations. In addition, some vortices are quite elongated. Both factors will have an impact on the suitability of the curvature centre method, as will be shown in Section 6.2.4.

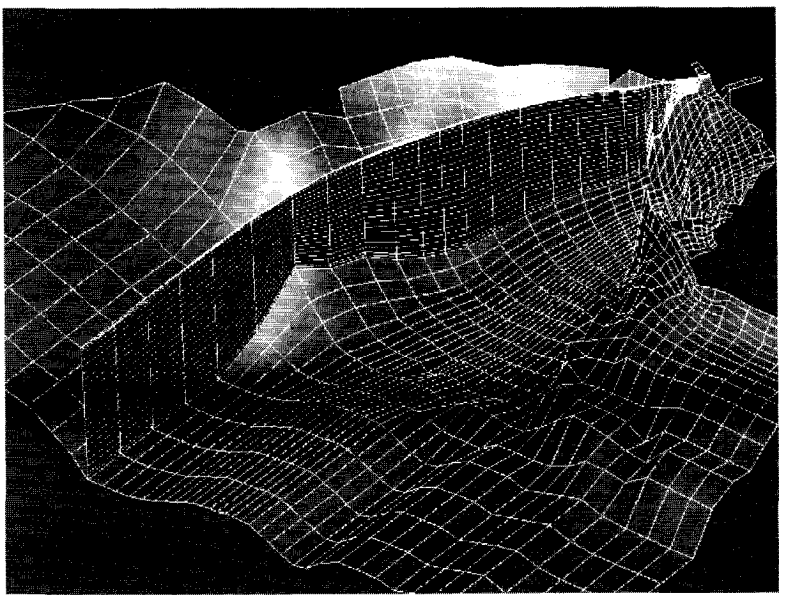
Figures 6.7 and 6.8 show four frames of a 200-frame particle animation made with PLANKTON. The particles are rendered as lines, scaled and coloured with the velocity magnitude. In addition, the sea floor geometry is rendered for orientation purposes. This geometry has been enlarged in the z-direction by a factor 250, as in [Hin, 1994]. The animation smoothly interpolates the view over all frames between the initial and final view points specified by the user. Initially, the view is from the east; later the view is rotated towards the northeast.

6.2.2 Vortex detection with scalar criteria

To see how well physical quantities indicate the presence of vortices, we use some of the quantities described in Section 4.1. From the velocity field, we derive vorticity ω, normalized helicity H_n, and second-largest eigenvalue of $\nabla \mathbf{v}$: λ_2. These quantities are mapped onto the same horizontal slice as in Figure 6.6, which results in Figures 6.9 and 6.10.



(a) Top view of a horizontal grid slice



(b) Side view from the northwest of the sea bed and a vertical grid slice

Figure 6.5: Bay of Gdańsk; computational grid

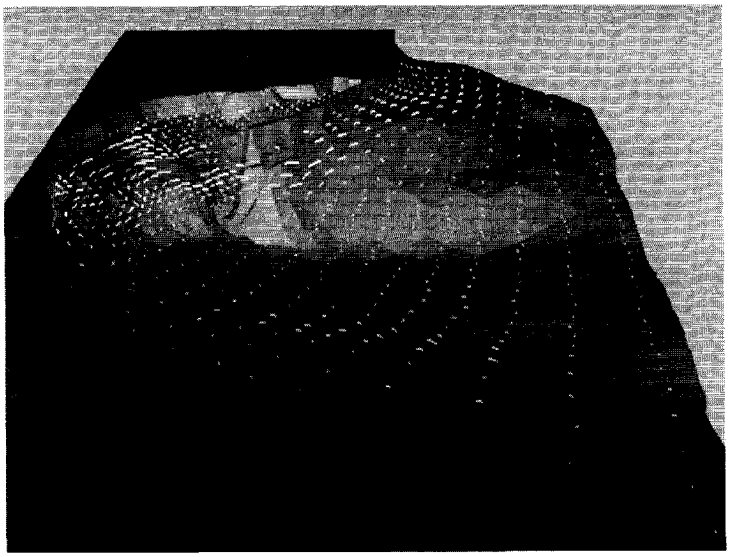
Figure 6.6: Bay of Gdańsk; flow pattern in horizontal grid slice

In these figures, we try to recognize the large obvious vortices from Figure 6.6, which serves as a reference figure. It can be seen that vorticity magnitude does not give any clue whatsoever as to where any vortices are located. The only maximum is in a small area close to the outflow of the Wisła river, hardly recognizable. The normalized helicity has many more peaks, but they do not bear a close resemblance to the vortices in the reference figure. Figure 6.10 shows the slice coloured with λ_2 and white iso-lines of $\lambda_2 = 0$, inside of which $\lambda_2 < 0$. The figure shows that λ_2 also has a small number of high peaks, and that some of the $\lambda_2 < 0$ regions seem to correspond to some vortices, but it is not a perfect correspondence. In conclusion, the scalar quantities do not give a satisfactory indication of where the vortices are located, especially the slow vortices.

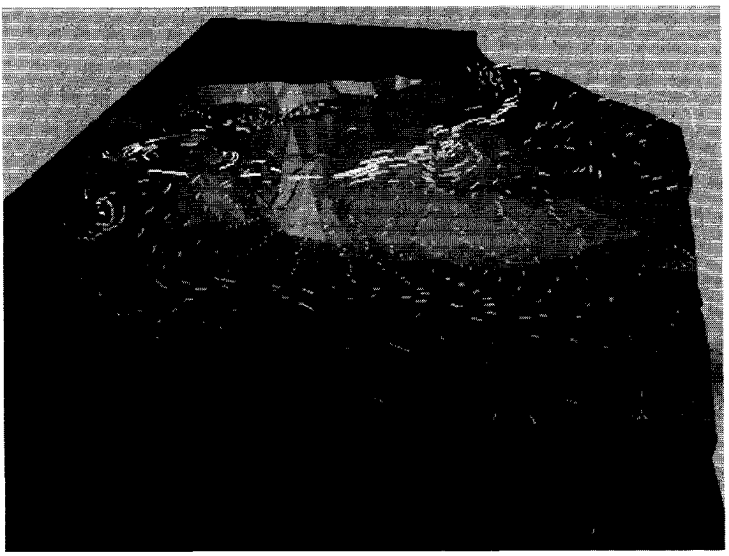
6.2.3 Vortex detection with critical points

We have also calculated critical points in order to find vortex cores. Figure 6.11 shows the critical points in the same data set, with different symbols for the various types.

The figure shows that most of the vortices have been captured, but not all of them:

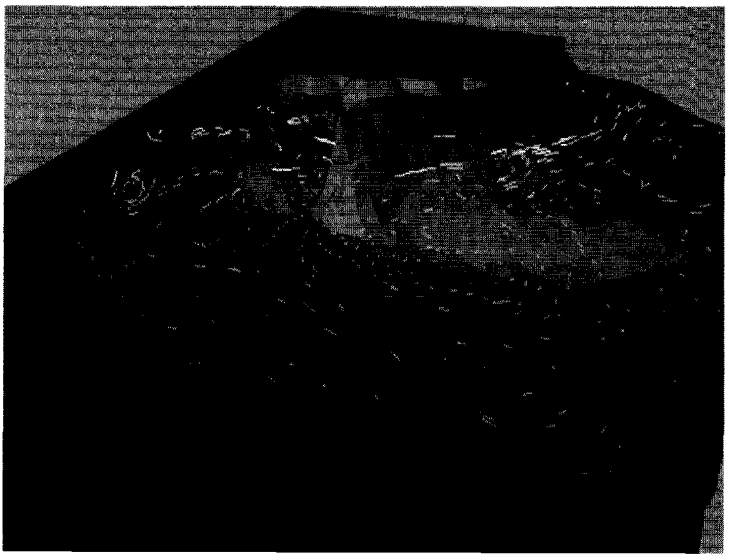


(a) Frame 0

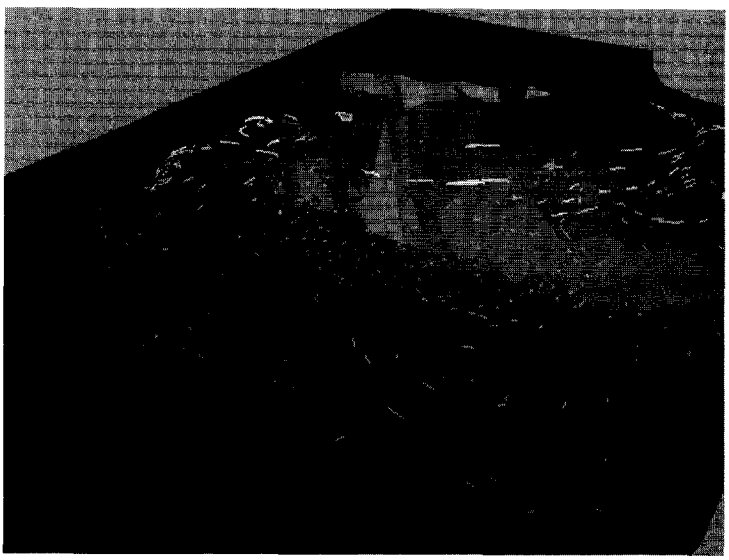


(b) Frame 40

Figure 6.7: Bay of Gdańsk; frames 0 and 40 of a PLANKTON animation

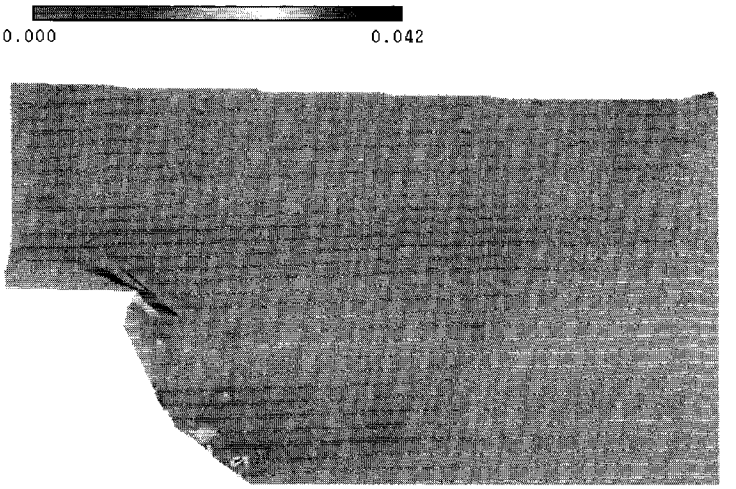


(a) Frame 80

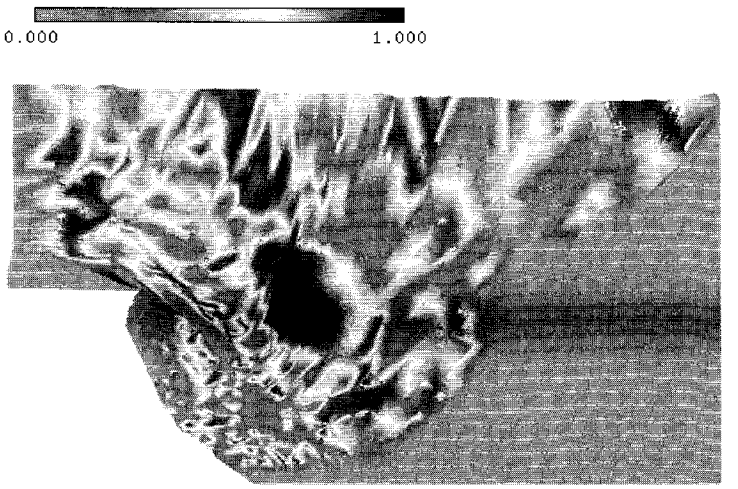


(b) Frame 120

Figure 6.8: Bay of Gdańsk; frames 80 and 120 of a PLANKTON animation



(a) Vorticity magnitude



(b) Normalized helicity

Figure 6.9: Bay of Gdańsk; horizontal slice coloured with ω and H_n

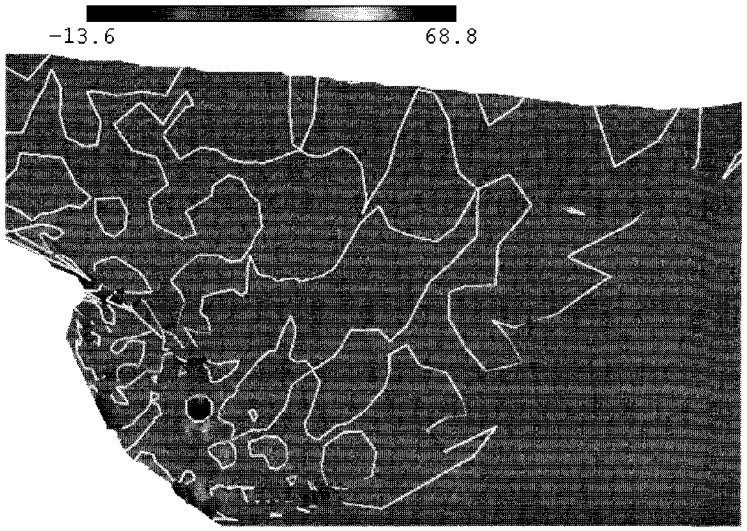


Figure 6.10: Bay of Gdańsk; horizontal slice coloured with λ_2. White lines indicate $\lambda_2 = 0$.

e.g. the vortex near $(340, 6120)$ has no focus but a node critical point, without any rotational component. Also, some elongated vortices have multiple critical points, e.g. the one near $(385, 6085)$ where the critical points are outside the intuitive rotation center. So, unfortunately, there is no one-to-one correspondence between critical points and vortices, which makes this method less suitable as the only method for finding vortex cores.

6.2.4 Vortex detection with curvature centres

Alternatively, we can apply geometric techniques for vortex detection, starting with the curvature centre technique described in Section 4.3. Figure 6.12 shows the curvature centre density (CCD) field as a coloured height field, along with the land geometry. The colour of the height field indicates the scalar value, to make it easier to distinguish the peaks.

The results are unsatisfactory, but in a different way than in the previous application. Here, the main problem is not too many false peaks, but too few; especially the slow vortices are notably absent.

The main causes are probably that those vortices are too slow and too elongated. Another cause could be that the curvilinear grid used in this data set causes the sampling density to be non-uniform throughout the grid. As the current implementation is limited to sampling at the grid nodes, the sampling density is lower in regions where

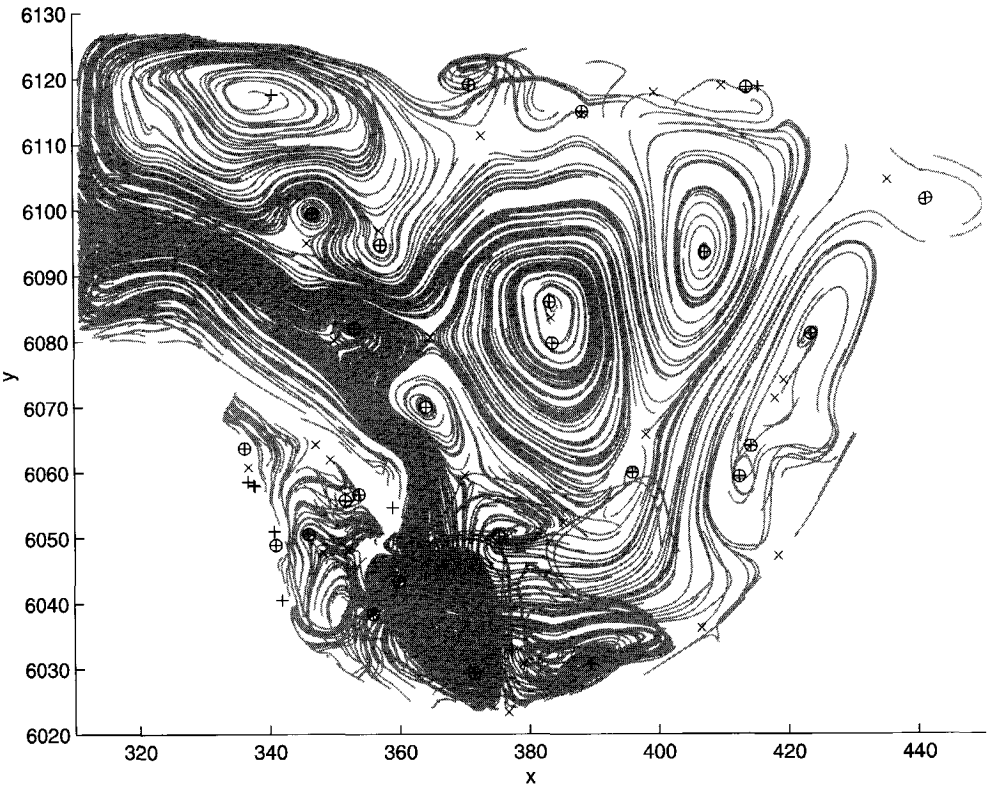


Figure 6.11: Critical points in the Bay of Gdańsk. 'o' are centres, '⊕' are foci, '+' are nodes, and '×' are saddle points.

the grid has larger cells, and vice versa. This could be compensated by making the sampling density inversely proportional to the local cell size, or by a complete resampling on a uniform grid. Yet, even with a uniformly spaced sampling, it still seems unlikely that the slow vortices will be captured, because the velocity magnitude is a dominant factor in the weight calculation (see Section 4.4).

In this case, applying the enhancements to the curvature centre technique suggested in Section 4.4 does not seem to be very useful. Thresholding can only reduce the number of peaks, not add any of the missing peaks. Filtering and supersampling only reduce the noise, which is not the main problem here, and of the alternative weighting schemes, the best alternative ($w = |\mathbf{v}|/r$) was used.

Figure 6.12: Bay of Gdańsk; height field of the curvature centre density

6.2.5 Vortex detection with the winding-angle method

As an alternative to the curvature centre method, we apply the winding-angle method (see Section 4.5) to the particle paths in Figure 6.6. As these particle paths are not long enough for the slowest vortices to make a sufficiently long loop, it is necessary to increase the number of time steps before applying the winding-angle method. Therefore, a maximum of 200 time steps of $\Delta t = 400s$ are calculated for each particle.

Then, particle paths are selected which satisfy the following criteria: winding-angle $\alpha_w \geq 1.5\pi$, absolute distance $d_{max,abs} \geq 15$ km, and relative distance $d_{max,rel} \geq 3 \cdot r_{est}$.

Next, the selected particle paths are clustered into vortices, by clustering their centre points. Particle paths of the same cluster are considered to belong to the same vortex.

Next, we perform quantification of the vortices, by calculating numerical attributes for them, some of which are listed in Table 6.2. Notice the differences between the largest and the smallest vortex (approximately a factor 20), and between the fastest and the slowest one (approximately a factor 15). There does not seem to be any correlation between the size and the rotation speed of the vortices.

Figure 6.13 shows how the numerical attributes are mapped to ellipse icons. The ellipses visualize the approximate size, shape, orientation, rotation speed, and rotation direction of the vortices. The ellipse axes are drawn in dashed lines, and the number of spokes indicates the strength of each vortex: the higher the number of spokes, the

Number of clusters	15
Number of CW vortices	5
Number of CCW vortices	10
Min. radius [km]	0.991
Max. radius [km]	21.2
Min. $\omega[s^{-1}]$	5.3810^{-5}
Max. $\omega[s^{-1}]$	8.9310^{-4}

Table 6.2: Some numerical attributes of the vortices in the Bay of Gdańsk.

faster the rotation. Arrow heads indicate the rotation direction. See also Colour Plate 6, where the rotation direction is also encoded in the colour of the ellipses: red ellipses rotate counterclockwise, green ellipses rotate clockwise.

This case has shown the use of particle tracing and vortex detection algorithms. The winding-angle method has again turned out to be better than the curvature centre method.

6.3 A transitional pipe flow

The application in this section concerns a direct numerical simulation (DNS) of a Poiseuille flow in a cylindrical pipe performed at the Laboratory for Aero- and Hydrodynamics at Delft University of Technology. Serving as a tool to explore the laminar-turbulent transition in pipe flow, the DNS tracks the spatial evolution of some local disturbance introduced from a wall area near the inflow [Ma *et al.*, 1998]. The disturbance, which is in the form of periodic suctioning/blowing (PSB), causes prominent streamwise vortex pairs to form, evolve, and break down. This process play an important role in the final transition to turbulent flow.

The simulation was performed on a 3D cylindrical grid containing $65 \times 17 \times 53$ nodes, which corresponds to a physical size of $-1.5\pi \leq x \leq 14.5\pi$, $0 \leq r \leq 1$, and $0 \leq \theta \leq 2\pi$, if (x, r, θ) are cylindrical coordinates. The x-coordinates are expressed as multiples of π for convenience in the numerical code. At $x = 0$, the disturbance is imposed. For the visualization of the vortices, we only use the parts between $0 \leq x \leq 5\pi$. The following quantities were calculated at each node: 3D velocity $\mathbf{v}$, vorticity magnitude ω, total pressure p, the second largest eigenvalue of the velocity gradient λ_2, its second invariant Q, and its discriminant Δ. More information on the physical background of this simulation may be found in [Ma *et al.*, 1998]. In this application, we use particle tracing and several vortex detection techniques.

6.3.1 Particle tracing

To visualize the global stream pattern, we apply particle tracing. We select nodes in a transverse plane at $x = 2\pi$ as the initial points of the particles, and trace in both

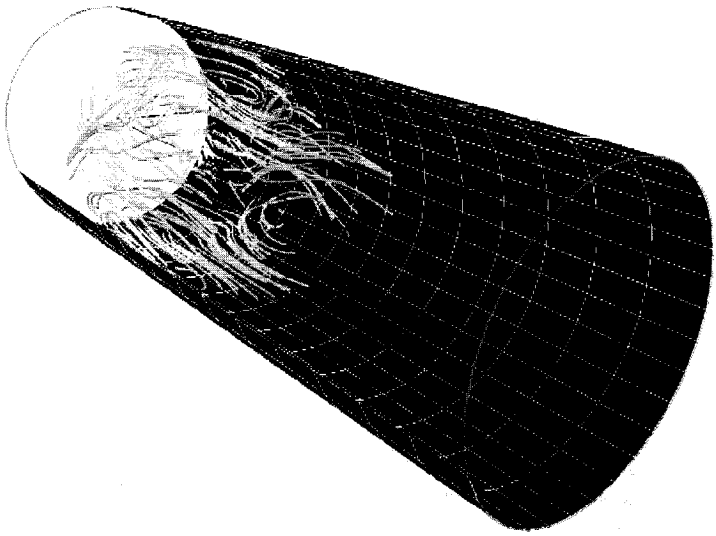
Figure 6.13: Bay of Gdańsk; vortices approximated by ellipses

directions. Since there is an extremely strong streamwise component, we do not use the physical velocity $\mathbf{v}$, but the disturbance velocity $\mathbf{v}'$ which is obtained by subtracting the mean velocity profile $\mathbf{v}' = \mathbf{v} - \bar{\mathbf{v}}$. Even so, there is still a strong streamwise component left.

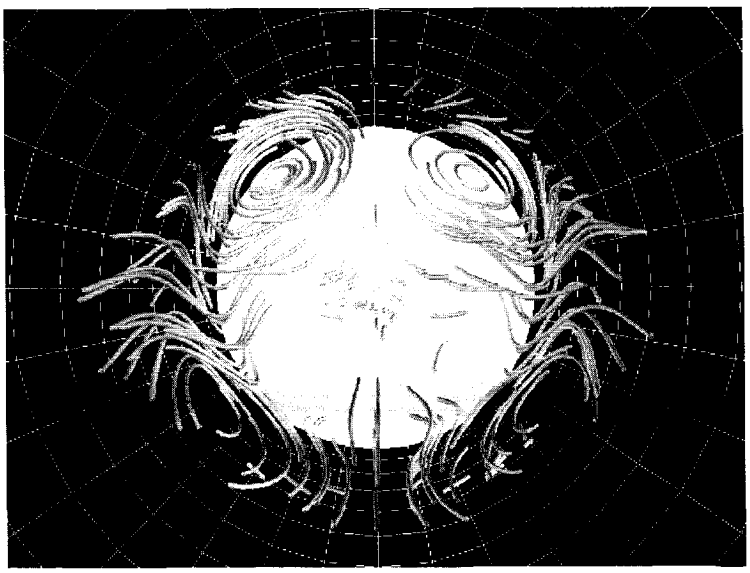
Figure 6.14a shows a side view of the resulting particle paths, rendered as solid tubes. As a depth cue, a part of the cylinder wall with the corresponding grid layer is shown. In this view, no vortical patterns are clearly visible, due to the strong streamwise component.

Figure 6.14b shows a better view, which in this case is a perspective view perpendicular to the cylinder axis. The perspective view clearly shows two vortex pairs, which are at an angle with the cylinder axis, both in the upper and lower half of the cylinder.

These figures show that this data set contains vortices which are not parallel to one of the natural coordinate axes. This makes it a little more difficult to apply 2D geometric vortex detection techniques, although the velocity field can still be projected



(a)



(b)

Figure 6.14: Side view (a) and front view (b) of pipe flow with particle traces released from slice $x = 2\pi$.

onto the x-, y-, or z-plane by just selecting two of the coordinates. This is shown in the following section.

6.3.2 Vortex detection with the winding-angle method

We have applied the winding-angle technique described in Section 4.5 to this case. Although the vortices are not completely perpendicular to the plane, the method still gives satisfactory results (see also [Sadarjoen *et al.*, 1998b]).

Figure 6.15 shows an example, where streamlines were selected with $\alpha_w > 1.3\pi$ and $d_{abs,max} = 2$.

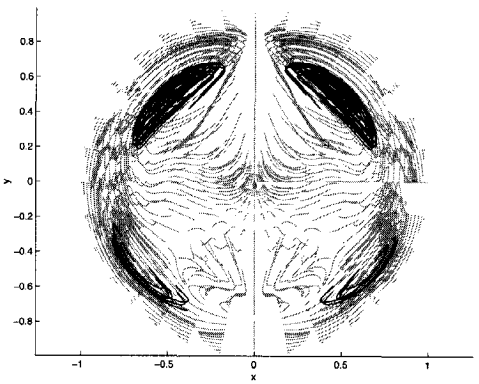


Figure 6.15: Pipe flow with selected streamlines

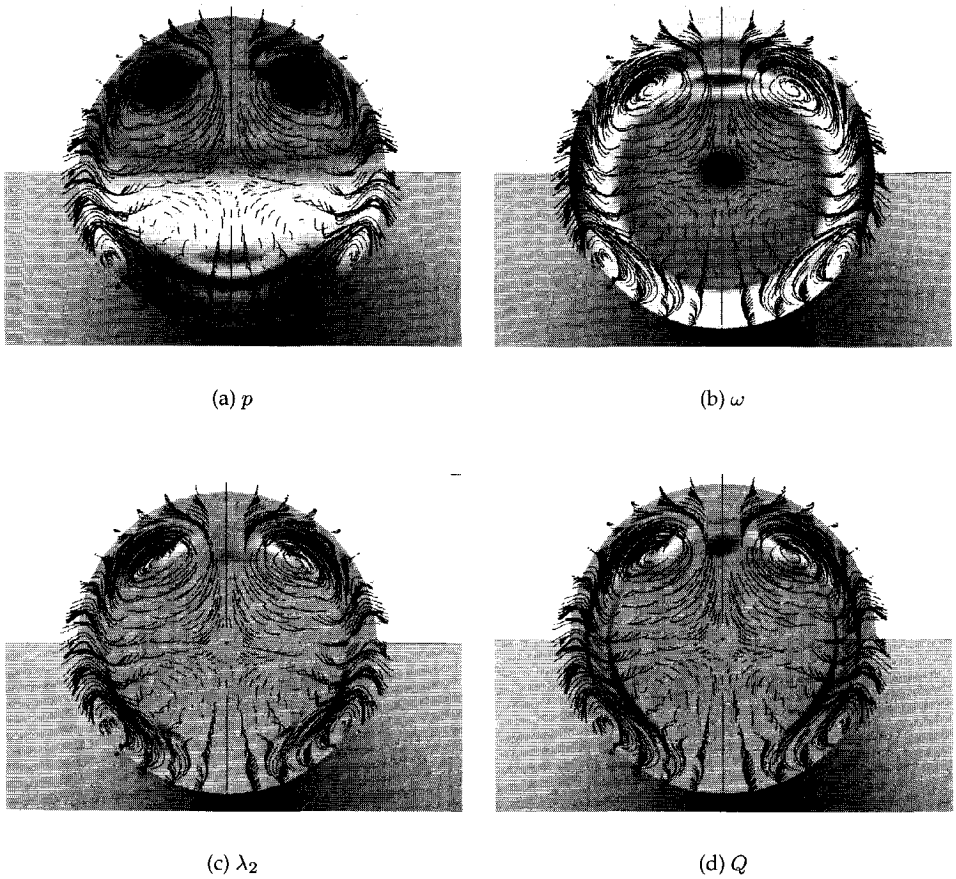
6.3.3 Vortex detection with scalar criteria

To see how well scalar quantities represent the vortices, which could be observed in the particle paths, we examine several scalar quantities. This is done below by using 2D transverse slices and 3D isosurfaces.

Figure 6.16 shows a transverse slice at $x = 2\pi$ coloured with four different quantities: pressure p, vorticity ω, second-largest eigenvalue of $\nabla \mathbf{v}$: λ_2, and second invariant of $\nabla \mathbf{v}$: Q. Particle paths released from the slice are shown for visual verification. See also Colour Plate 7 for a colour version of Figure 6.16a.

The figure shows that low pressure and high vorticity ω are not good vortex indicators in this case, because they have no maxima where the vortices actually are, but near the cylinder axis (pressure), and near the walls (vorticity). The latter could be attributed to the no-slip condition which holds at the walls. In contrast, highly negative λ_2 indicates the presence of vortices very well, as does highly positive second invariant Q. Q is almost equivalent to $-\lambda_2$, but a little smoother.

3D views of the same quantities are shown in Figures 6.17–6.19 which depict isosurfaces of ω, λ_2, and Q. These figures confirm the observations in 2D that ω does



(a) p (b) ω

(c) λ_2 (d) Q

Figure 6.16: Pipe flow with particle paths and slices coloured with various scalar quantities

not represent the vortices very well. Areas of highly negative λ_2 and high Q seen to correspond to the vortices very well, with Q being a little smoother than λ_2.

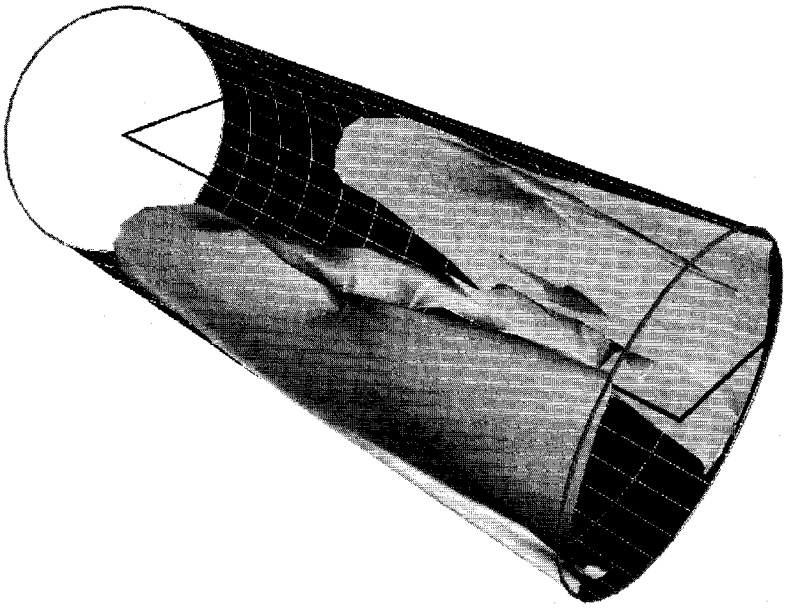


Figure 6.17: isosurface of high ω

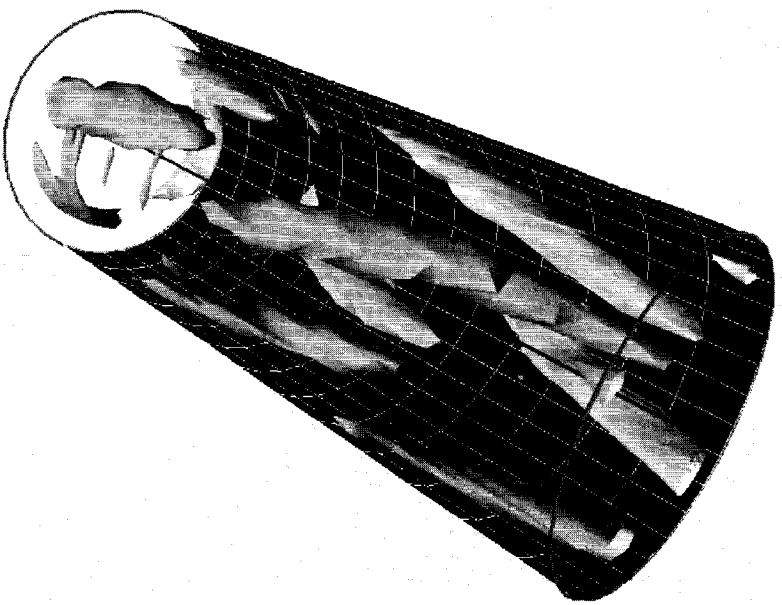


Figure 6.18: isosurface of highly negative λ_2

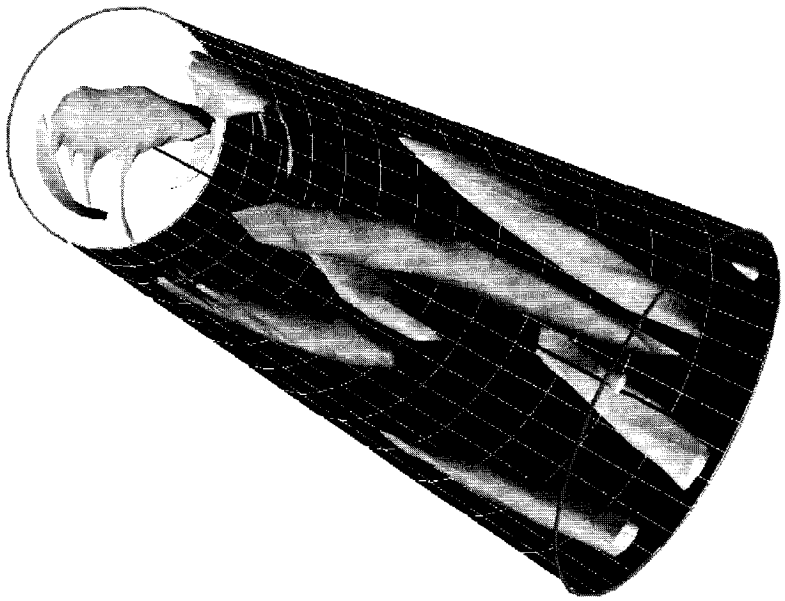


Figure 6.19: isosurface of high Q

6.3.4 Vortex detection with selective and iconic techniques

In order to verify the assumption that the vortices are represented by any of the previous scalar criteria, we trace particles in just those areas that satisfy these criteria. The problem of finding the initial points is solved by using the selection techniques developed by Van Walsum [van Walsum & Post, 1994]. These techniques allow the user to select grid nodes in a data set, where the data satisfies a certain condition. Here, the selection criterion is $Q > 0.8 \cdot Q_{max}$. From these nodes, it is possible to start particle tracing in both the forward and backward directions. A result is shown in Figure 6.20.

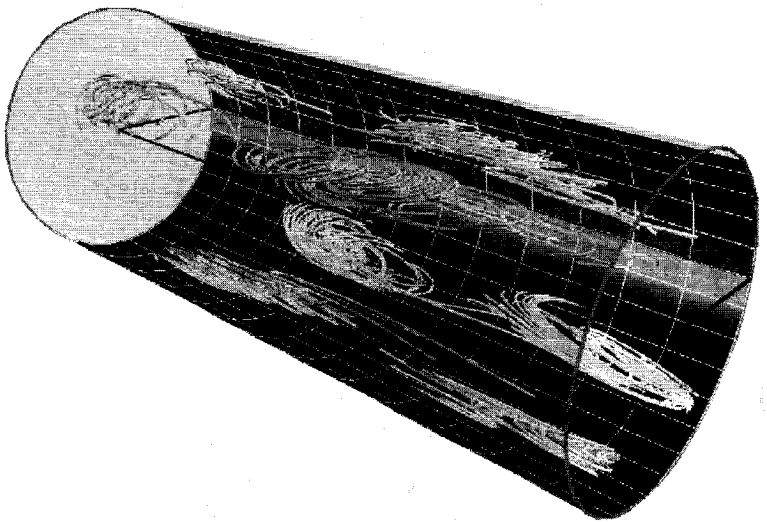


Figure 6.20: Selective streamlines

In order to compare the particle paths with the vortical regions indicated by high Q, Figure 6.21 shows an image of these streamlines combined with semi-transparent isosurfaces of a $Q = 0.8 \cdot Q_{max}$. See also Colour Plate 8 for a colour version.

These regions of selected points can also be used to apply iconic visualization and feature extraction techniques. Figure 6.22 shows an example, where ellipsoid icons have been fitted through the selected grid nodes where $Q > 0.8 \cdot Q_{max}$. This figure clearly shows that the vortex pairs are at an angle with the cylinder axis. The angles could be determined from the eigenvector directions of the ellipsoids. See the top image on the back cover for a colour version.

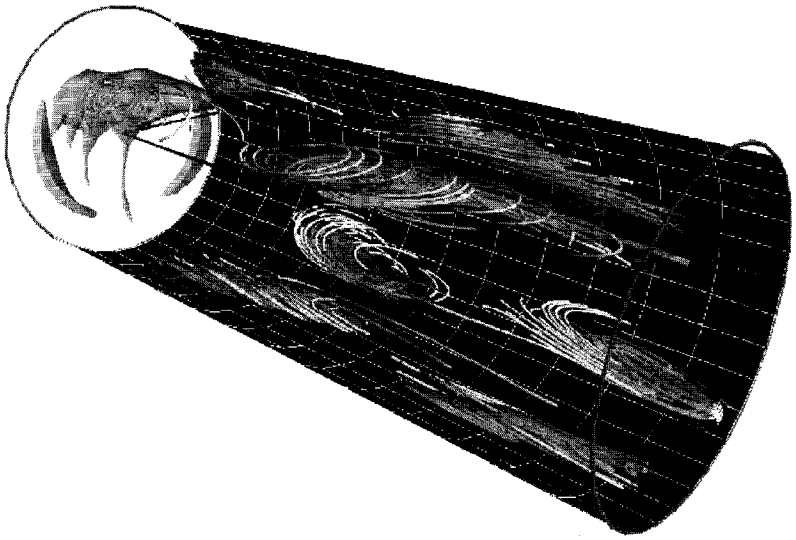


Figure 6.21: Selective streamlines and isosurfaces of Q

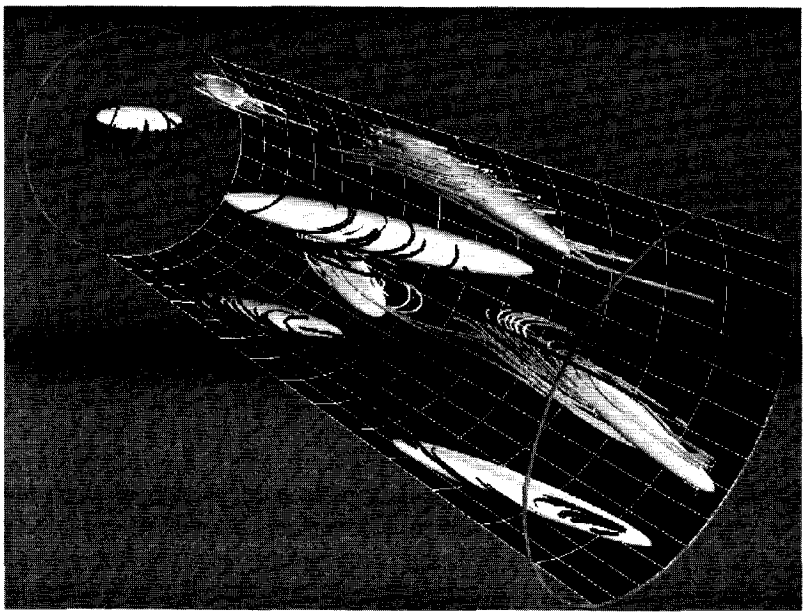
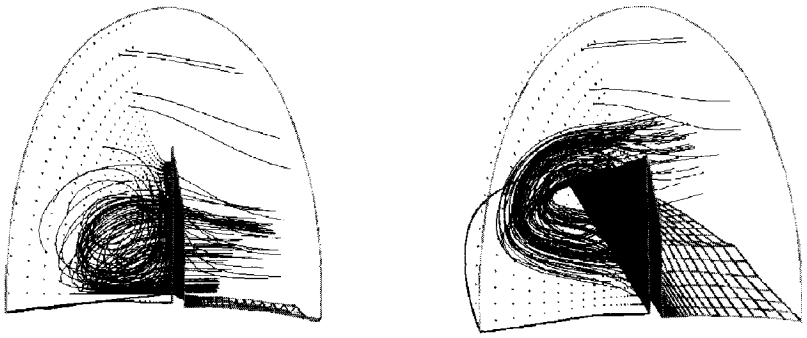


Figure 6.22: Selective streamlines and ellipoid icons

6.4 The Delta-Wing aircraft

The data set used here is the Delta-Wing data set made available by NASA Ames Research Center. It models the flow around a Delta-Wing aircraft at a 40 degrees angle of attack [Ekaterinis & Schiff, 1990]. The features in this data set include vortices on one side of the wing. The original files contain the following quantities: density, momentum, and stagnation, which are defined on a $56 \times 54 \times 70$ structured curvilinear grid. In this application, we use particle tracing, vortex detection techniques, and deformable surfaces.

6.4.1 Particle tracing



(a) Projected to the xz-plane

(b) Best view on vortex

Figure 6.23: Delta-Wing with particle traces

The figures show that the vortices are not parallel with any of the natural axes, or even to any of the i, j, and k-planes in computational space. This makes it difficult to apply one of the 2D vortex detection techniques. Therefore, we apply a 3D approach for vortex extraction, namely deformable surfaces (see Chapter 5).

6.4.2 Vortex detection using deformable surfaces

As in Section 6.3.1, the visualization with particle tracing has shown that the vortices in this data set are not aligned with one of the natural coordinate axes. In this case, we may apply deformable surfaces (see Chapter 5, as they have the advantage that they are inherently 3D, rather than limited to 2D projections. To detect vortices, the deformable surface technique can be applied with two types of deformation criteria: scalar criteria and vector criteria.

Deformable surfaces with scalar criteria

The extraction of vortex tubes proceeds in two steps: the first step uses a selection criterion to find the vortex core, which is then used to initialize a deformable surface. The second step uses a deformation criterion to find the outer surface of the vortex tube. For both steps, we can distinguish two types of criteria: scalar criteria and vector criteria. The first type will be covered in the current section, the other in the next section.

In accordance with the scheme in Section 5.1, the following steps were performed:

- Region selection: the quantity used here is pressure p. The selection criterion is minimal pressure, $p < c \cdot \min(p)$, where c is a user-definable parameter. The result is shown in Figure 6.24a, where the selected points are again shown as cross marks.
- Initial Surface Creation: in the selected regions, integral attributes are calculated, which results in the ellipsoid shown in Figure 6.24b. The ratios of the ellipsoid axes result in a 1D object, the cylinder shown in Figure 6.24c.
- Surface deformation: for the deformation stage, again p is used. The result is therefore a local isosurface of p.

Figure 6.24d shows the resulting vortex tube after one iteration with the single-step step size determination method. It consists of 180 triangles. This surface takes 3.1 s to generate on an SGI Indy, which makes it suitable for use as an interactive tool.

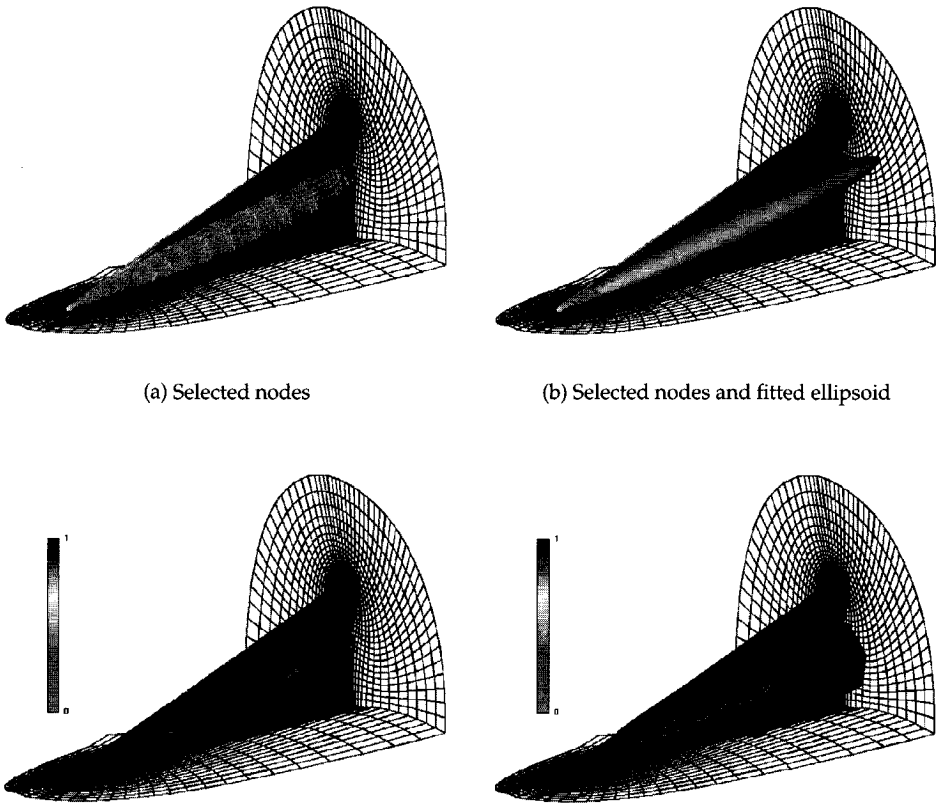
Deformable surfaces with vector criteria

We also used other criteria, based on additional (vector) quantities, to extract the vortex tube from the same data set. Again following the scheme in Section 5.1, these steps were performed:

- Region Selection: the vortex tube extraction now uses additional physical quantities, which have been derived from the original quantities in the data set: the vorticity vector field ω and its magnitude $|\omega|$. These were also used in [Banks & Singer, 1994; Villasenor & Vincent, 1992]. The criterion used for the initialization is: low pressure and high vorticity magnitude: $p < c_1 \cdot \min(p)$ and $\omega > c_2 \cdot \max(\omega)$ where c_1 and c_2 are user-definable parameters.
- Initial Surface Creation: in the selected region, integral attributes are calculated which result in an ellipsoid. The ratios of the ellipsoid axes result in a 1D object, the cylinder shown in Figure 6.26a. The colour indicates cost function, where red is highest and blue is lowest.
- Surface Deformation: the deformation criterion uses the angle between the vorticity vector ω_c at the vortex core and the vorticity vector ω_s at a node $\mathbf{n}$ of the deformable surface: $\alpha = \angle(\omega_s, \omega_c)$. Figure 6.25 shows that $\alpha = 0$ at the core.

The algorithm makes the surface grow in the direction where this $\alpha = 90°$, or as high as possible but lower than 90°. Since the cost function is defined as $\cos\alpha$, and $0 \leq \alpha \leq 90°$, minimizing the cost function maximizes the angle:

$$C(\mathbf{x}_n) = \cos(\angle(\omega_s, \omega_c)) \tag{6.1}$$



(a) Selected nodes

(b) Selected nodes and fitted ellipsoid

(c) Initial vortex tube

(d) Final vortex tube

Figure 6.24: Delta-Wing; extraction of a vortex tube with scalar deformation criteria

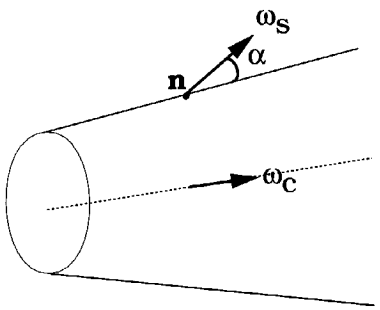
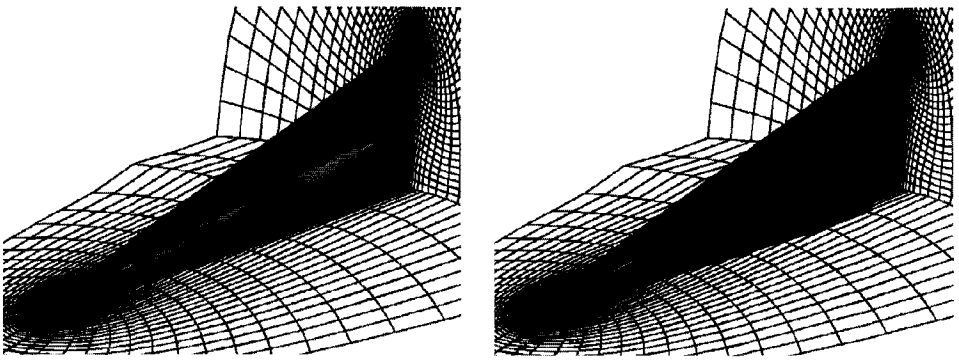


Figure 6.25: Delta-Wing; vortex-finding with vector criteria



(a) Initial vortex tube

(b) Final vortex tube

Figure 6.26: Extraction of a vortex tube with vector deformation criteria. Colour indicates the value of the cost function C (red = high, blue = low).

Figure 6.26b shows the final vortex tube after 16 iterations with the multi-step iteration method, each of which takes about 3 s. The surface consists of 260 triangles; the colour indicates the cost function.

This application shows that deformable surfaces can be effectively employed in cases where other, 2D techniques fail, since deformable surfaces can employ both scalar and vector criteria.

Chapter 7

Conclusions and future work

Starkt, evigt och starkt [1]
är vågornas väldiga tåg,
och stark av det eviga havet
var mjuk förgänglig våg.

The preceding chapters in this thesis describe techniques for extracting and visualizing *features*, i.e. interesting and characteristic phenomena in a data set, using *geometries*, which are curves, surfaces and solids. Three types of techniques have been covered: particle tracing, vortex detection, and deformable surfaces. In this chapter, we draw conclusions and give directions for future research.

7.1 Conclusions

Particle tracing

In Chapter 3, we described particle tracing, a technique to visualize velocity fields by simulating the release of massless particles and calculating their trajectories. We concentrated on particle tracing in various types of special grids: structured curvilinear grids, σ-transformed grids, and unstructured grids. In these grids, point location, the process of finding which grid cell contains a given point, is more difficult.

In structured curvilinear grids consisting of hexahedral cells, decomposition of the cells into 5 tetrahedra usually works well. In σ-transformed grids, a frequently used type of grid in hydrodynamic applications, the 5-decomposition often fails. In that case, applying a different decomposition, viz. into 6 tetrahedra, has proven to be a robust alternative. In unstructured grids consisting of hexahedral or tetrahedral cells, it is only possible to perform particle tracing after the addition of connectivity information, because it is not included. For this purpose, a library called `CNX-lib` was developed.

It can be concluded from the examples in Chapter 3 and from the applications in Chapter 6, that particle tracing is now possible in these special grid types. Particle tracing was found to be a useful technique for exploring velocity fields, both when used

[1] Strong, eternal and strong // is the huge row of the waves // and strong due to the eternal sea // was the soft and fleeting wave. (Karin Boye: *Havet*)

globally and selectively. When used globally, it gives a good overview of the data, e.g. by releasing particles at every grid node of a grid slice. If globally releasing particles leads to too many streamlines and cluttering, techniques such as in [Turk & Banks, 1996; Mao *et al.*, 1998] may be used to distribute them uniformly. When particle tracing is used selectively, selection techniques are used to select grid nodes which satisfy a certain criterion, and particles or stream lines are released only there. This prevents cluttering and leads to a clearer view of the data. In both cases, particle tracing may be used as a 'reference' technique for verifying the results of vortex extraction techniques.

Vortex detection

In Chapter 4, vortex detection was described. Two types of vortex detection criteria were covered, physical criteria and geometric criteria.

Physical criteria are based on physical quantities evaluated at many points in the flow field, or in infinitesimal regions around these points. Examples are low pressure, high vorticity and high helicity. Physical criteria often turned out to be unreliable, because they are based on point samples, while vortices are regional phenomena.

Geometric criteria are based on the shape properties of 2D flow curves, typically projected stream lines or path lines. Two types of geometric vortex detection methods were given.

The first method, called the Curvature Center Density (CCD) method, determines the curvature centre for all points in a grid node, by calculating the centre of the osculating circle of the streamline through that node. Regions having a high density of curvature centres are assumed to contain vortices. Unfortunately, the results of this method turned out to be disappointing, due to its sensitivity to imperfect circular shapes of streamlines and to the magnitude of the velocity. As a result, deformed vortices and weak vortices were usually not found.

The other method, called the winding-angle method, basically counts the number of loops made by a flow curve. The method does not depend on the exact shape of the curve nor on the velocity magnitude, as long as a loop is made. In addition, it allows for quantification of the detected vortices. Numeric attributes can be determined, and mapped to ellipses, which approximate the shape, size, and orientation of the vortices, and which can also be used to visualize the strength of a vortex. This method gave much better results than the CCD method.

It can be concluded that geometric criteria for vortex detection are better in some cases than physical criteria; especially the winding-angle method is capable of detecting non-circular and weak (slowly rotating) vortices not found by physical criteria.

Deformable surfaces

In Chapter 5, we described deformable surfaces, surfaces which start from some initial shape and are deformed to the shape of a feature. We described mechanisms for initializing, deforming, and refining deformable surfaces.

An initial surface is created by first selecting grid nodes in a region of interest, and fitting an ellipsoid to the selected nodes. The shape of the ellipsoid is then used to determine the initial surface. Surface deformation is accomplished by displacing the nodes in an iterative way. We used two mechanisms for this: firstly, a multi-step method, which displaces the nodes over small distances at a time, and requires many iteration steps for the growth. Secondly, a single-step method, which displaces nodes over large distances, and requires only one iteration step for the growth. For surface refinement, several mechanisms were investigated, based on subdivision of faces into two or four triangles. We have shown three example applications of deformable surfaces: extracting local isosurfaces, extracting recirculation zones, and extracting vortices.

The examples in Chapters 5 and 6 have demonstrated that deformable surfaces are a useful and generic tool for extracting features that cannot always be defined using a global scalar field, like vortex tubes and recirculation zones. Another advantage is that they are capable of using scalar and vector criteria.

7.2 Future work

Future work on particle tracing could include research on algorithms which work on yet other types of grids. Higher-accuracy integration and interpolation schemes combined with physically-based particle tracing methods [Kenwright & Mallinson, 1992b] could allow for even more accurate visualizations. When an extremely large number of particles is used, acceleration could be obtained by optimization, e.g. by using caching mechanisms for point location, or by parallelization.

Future work on vortex detection includes the extension and generalization of the winding-angle technique to 3D. As the winding-angle is only defined in relation to a (projection) plane, one of the challenges in 3D will be to find a global projection plane, or if possible, a locally varying projection plane at each sample point of the 3D curve. This is especially a challenge when there is a strong forward velocity component.

In the curvature centre method for vortex detection, a clustering technique could be applied to detect vortex basins, by recognizing a limited number of vortex cores and associating nearby points with them.

The computational costs of the winding-angle method could be reduced by releasing streamlines not globally, but locally in regions which are pre-selected using physical criteria. However, a condition for this to work, is that the pre-selection criteria must give a superset of the actual vortices, as the winding-angle technique can only reduce the set.

Regarding deformable surfaces, there are several themes for future work: the single-step method could be improved to become less sensitive to non-monotonic fields with high gradients, by implementing step size control to prevent nodes from jumping in all directions. To this end, progress coordination of adjacent nodes could also be useful. The idea is that for each node, we keep track of how much 'progress' it has made since the deformation started. Also, it is estimated how much progress would be made if a

node would be displaced over a certain distance. Then, if at some instance, a node has made significantly more progress than its neighbours, its movement could be 'frozen' in one or more subsequent steps, so the neighbours would get a chance to 'synchronize'.

Currently, our deformable surfaces have a fixed topology. More flexible types of surfaces worth investigating would be topologically adaptable surfaces, capable of splitting and merging, as described in [Metaxas, 1997], and surfaces consisting of only points, not connected by polygons, such as the ones described in [Witkin & Heckbert, 1994].

Also, more complex optimization schemes could be investigated, such as simulated annealing and dynamic programming. Last but not least, it will be interesting to find more application-specific selection and deformation criteria for using deformable surfaces to extract other kinds of features. These are not necessarily limited to fluid flows; fractures in soil mechanics might also be a potential application area for deformable surfaces. To achieve the best results, new criteria should be developed in collaboration with experts from the relevant fields.

Bibliography

ALBERTELLI, G., & CRAWFIS, R.A. 1997. Efficient Subdivision of Finite-Element Datasets into Consistent Tetrahedra. *Pages 213–219 of:* YAGEL, R., & HAGEN, H. (eds), *Proc. Visualization '97*. ACM Press.

AVS. 1993. *AVS Module Reference*. Release 5 edn. Advanced Visual Systems, Inc., Waltham MA.

BANKS, D.C., & SINGER, B.A. 1994. Vortex tubes in turbulent flows: identification, representation, reconstruction. *Pages 132–139 of:* BERGERON, R.D., & KAUFMAN, A.E. (eds), *Proc. Visualization '94*. IEEE Computer Society Press.

BATCHELOR, G.K. 1967–1994. *An Introduction to Fluid Dynamics*. Cambridge: Cambridge University Press.

BLOOMENTHAL, J. (ed). 1997. *Introduction to implicit surfaces*. Morgan Kaufmann.

DE BOER, A.J. 1998. *Reconstructie en uitbreiding van Plankton in AVS/Express*. M.Sc. thesis, Delft University of Technology.

VAN DEN BROEK, S.P., REINDERS, K.F.J., DONDERWINKEL, M., & PETERS, M.J. 1998. Volume conduction effects in EEG and MEG. *Electroencephalography and Clinical Neurophysiology*, **106**(6), 522–534.

BUNING, P. 1989. Numerical Algorithms in CFD Post-Processing. *In: Computer Graphics and Flow Visualization in Computational Fluid Dynamics*. Lecture Series 1989-07. Von Kármán Institute for Fluid Dynamics, Brussels, Belgium.

CASTLEMAN, K.R. 1996. *Digital Image Processing*. Englewood Cliffs, NJ: Prentice-Hall.

CHEIN, R.R. 1990. Numerical modelling of unsteady backward-facing step flow. *Journal of the Chinese Institute of Engineers*, **13**(1), 69–82.

CHONG, M.S., PERRY, A.E., & CANTWELL, B.J. 1990. A General Classification of Three-Dimensional Flow Fields. *Phys. Fluids A*, **2**(5), 765–777.

DELMARCELLE, T., & HESSELINK, L. 1993. Visualizing Second-Order Tensor Fields with Hyperstreamlines. *IEEE Computer Graphics and Applications*, **13**(4), 25–33.

EKATERINIS, J.A., & SCHIFF, L.B. 1990 (January). Vortical Flows over Delta Wings and Numerical Prediction of Vortex Breakdown. *In: AIAA Aerospace Sciences Conference.* AIAA Paper, nos. 90–0102. http://science.nas.nasa.gov/Software/DataSets.

FARIN, G. 1990. *Curves and Surfaces for Computer Aided Geometric Design.* Academic Press.

FUHRMANN, A., & GRÖLLER, E. 1998. Real-Time Techniques for 3D flow visualization. *Pages 305–312 of:* EBERT, D., HAGEN, H., & RUSHMEIER, H. (eds), *Proc. Visualization '98.* IEEE Computer Society Press.

GROSSO, R., & ERTL, T. 1998. Progressive Iso-surface extraction from hierarchical 3D meshes. *Pages C125–C135 of:* FERREIRA, N., & GÖBEL, M. (eds), *Proc. Eurographics '98.* Computer Graphics Forum, vol. 17 (3).

HAMANN, B., WU, D., & MOORHEAD, R.J. 1995. On particle path generation based on quadrilinear interpolation and Bernstein-Bezier polynomials. *IEEE Transactions on Visualization and Computer Graphics,* **1**(3), 210–217.

HELMAN, J., & HESSELINK, L. 1991. Visualizing Vector Field Topology in Fluid Flows. *IEEE Computer Graphics and Applications,* **14**(5), 36–46.

HIN, A.J.S. 1994. *Visualization of Turbulent Flow.* Ph.D. thesis, Delft University of Technology.

HIN, A.J.S., & POST, F.H. 1993. Visualization of Turbulent Flow with Particles. *Pages 46–52 of:* NIELSON, G.M., & BERGERON, R.D. (eds), *Proceedings Visualization '93.* IEEE Computer Society Press.

HULTQUIST, J.P.M. 1992. Constructing Stream Surfaces in Steady 3D Vector Fields. *Pages 171–178 of:* KAUFMAN, A.E., & NIELSON, G.M. (eds), *Proceedings Visualization '92.* IEEE Computer Society Press, Los Alamitos, CA.

HUNT, J.C.R., WRAY, A.A., & MOIN, A.A. 1988. Eddies, Streams and Convergences Zones in Turbulent Flows. *Pages 193–208 of: Center for Turbulence Research Proceedings of Summer Program.*

JEONG, J., & HUSSAIN, F. 1995. On the identification of a vortex. *Journal of Fluid Mechanics,* **285**, 69–94.

JESPERSEN, D.C., & LEVIT, C. 1991. *Numerical simulation of flow past a tapered cylinder.* Tech. rept. AIAA 91-0751. NASA Ames Research Center, Reno, NV. 29th AIAA Aerospace Sciences Meeting and Exhibit.

KASS, M., WITKIN, A., & TERZOPOULOS, D. 1988. Snakes: Active Contour Models. *International Journal of Computer Vision,* **1**(4), 321–331.

KENWRIGHT, D., & HAIMES, R. 1997. Vortex Identification - Applications in Aerodynamics: A Case Study. *Pages 413–416 of:* YAGEL, R., & HAGEN, H. (eds), *Proc. Visualization '97.* ACM Press.

KENWRIGHT, D.N., & LANE, D.A. 1995. Optimization of Time-Dependent Particle Tracing Using Tetrahedral Decomposition. *Pages 321–328 of:* NIELSON, G.M., & SILVER, D. (eds), *Proc. Visualization '95.* IEEE Computer Society Press.

KENWRIGHT, D.N., & MALLINSON, G.D. 1992a. A 3-D Streamline Tracking Algorithm Using Dual Stream Functions. *Pages 62–68 of:* KAUFMAN, A.E., & NIELSON, G.M. (eds), *Proceedings Visualization '92.* IEEE Computer Society Press, Los Alamitos, CA.

KENWRIGHT, D.N., & MALLINSON, G.D. 1992b. A 3-D Streamline Tracking Algorithm Using Dual Stream Functions. *Pages 62–68 of:* KAUFMAN, A.E., & NIELSON, G.M. (eds), *Proc. Visualization '92.* IEEE CS Press.

DE LEEUW, W.C. 1997. *Presentation and Exploration of Flow Data.* Ph.D. thesis, Delft University of Technology.

DE LEEUW, W.C., & POST, F.H. 1995. A Statistical View on Vector Fields. *Pages 53–62 of:* GÖBEL, M., MÜLLER, H., & URBAN, B. (eds), *Visualization in Scientific Computing.* Eurographics. Wien: Springer-Verlag.

LOBREGT, S., & VIERGEVER, M.A. 1995. A Discrete Dynamic Contour Model. *IEEE Transactions on Medical Imaging*, **14**(1), 12–24.

LORENSEN, W.E., & CLINE, H.E. 1987. Marching Cubes: a high resolution 3D surface construction algorithm. *Computer Graphics*, **21**(4), 163–169. Proc. Siggraph '87.

MA, B., VAN DOORNE, C.W.H., ZHANG, Z., & NIEUWSTADT, F.T.M. 1998. On the evolution of wall-imposed periodic disturbance in pipe Poiseuille flow at Re=3000. *submitted to: Journal of Fluid Mechanics.*

MAO, X., HATANAKA, Y., HIGASHIDA, H., & IMAMIYA, A. 1998. Image-Guided Streamline Placement on Curvilinear Grid Surfaces. *Pages 135–142 of:* EBERT, D., HAGEN, H., & RUSHMEIER, H. (eds), *Proc. Visualization '98.* IEEE Computer Society Press.

MCINERNEY, T., & TERZOPOULOS, D. 1996. Deformable models in medical image analysis: a survey. *Medical Image Analysis*, 1(2), 91–108.

MEIJER, D.G. 1995. *Lock Approach of Second Ship Lock at Lith. Scale model investigation and DELFT3D-calculations.* Tech. rept. WL | Delft Hydraulics. In Dutch.

METAXAS, D.N. 1997. *Physics-based deformable Models; Applications to computer vision, graphics and medical imaging.* Kluwer Academic Publishers.

MILLER, J.V. 1990. *On GDMs: Geometrically Deformed Models for the Extraction of Closed Shapes from Volume Data.* M.Sc. thesis, Rensselaer Polytechnic Institute, Troy, New York.

MILLER, J.V., BREEN, D.E., LORENSEN, W.E., O'BARA, R.M., & WOZNY, M.J. 1991. Geometrically Deformed Models: A Method for Extracting Closed Geometric Models from Volume Data. *Pages 217–226 of: Proc. SIGGRAPH 1991.* ACM.

MITCHELL, W.F. 1991. Adaptive refinement for arbitrary finite-element spaces with hierarchical bases. *J. of Computational and Applied Mathematics,* **36**, 65–78.

MÜLLER, K., RIST, U., & WAGNER, S. 1998. Enhanced visualization of late-stage transitional structures using vortex identification and automatic feature extraction. *Pages 786–791 of: Proc. ECCOMAS'98.* the European Community on Computational Methods in Applied Sciences.

MYNETT, A.E., WESSELING, P., SEGAL, A., & KASSELS, C.G.M. 1991. The ISNaS incompressible Navier-Stokes solver: invariant discretization. *Applied Scientific Research,* **48**, 175–191.

MYNETT, A.E., SADARJOEN, I.A., & HIN, A.J.S. 1995. Turbulent Flow Visualization in Computational and Experimental Hydraulics. *Pages 388–391 of:* NIELSON, G.M., & SILVER, D. (eds), *Proceedings Visualization '95.* IEEE Computer Society Press.

PAGENDARM, H.-G., & POST, F.H. 1997. Studies in Comparative Visualization of Flow Features. *Chap. 9, pages 211–227 of:* NIELSON, G.M., HAGEN, H., & MÜLLER, H. (eds), *Scientific Visualization: Overviews, Methodologies, Techniques.* IEEE Computer Society Press.

PAGENDARM, H.-G., & WALTER, B. 1994. Feature Detection from Vector Quantities in a Numerically Simulated Hypersonic Flow Field in Combination with Experimental Flow Visualization. *Pages 117–123 of:* BERGERON, R.D., & KAUFMAN, A.E. (eds), *Proceedings Visualization '94.* IEEE Computer Society Press, Los Alamitos, CA.

PERRY, A.E., & CHONG, M.S. 1987. A Description of Eddying Motions and Flow Patterns using Critical-Point Concepts. *Ann.Rev. Fluid Mech.,* **19**, 125–155.

PORTELA, L.M. 1997. *On the Identification and Classification of Vortices.* Ph.D. thesis, Stanford University, School of Mechanical Engineering.

POST, F.H., & VAN WALSUM, T. 1993. Fluid Flow Visualization. *Pages 1–40 of:* HAGEN, H., MÜLLER, H., & NIELSON, G.M. (eds), *Focus on scientific visualization.* Springer-Verlag. Papers presented at the first Dagstuhl Seminar on Scientific Visualization, August 1991.

POST, F.H., DE LEEUW, W.C., SADARJOEN, I.A., REINDERS, F., & VAN WALSUM, T. 1999. Global, Geometric, and Feature-Based Techniques for Vector Field Visualization. *Future Generation Computer Systems (Special Issue: Scientific Visualization)*, **15**(1), 87–98.

PRESS, W.H., TEUKOLSKY, S.A., VETTERLING, W.T., & FLANNERY, B.P. 1992. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press.

REINDERS, F., POST, F.H., & SPOELDER, H.J.W. 1997. Feature Extraction from Pioneer Venus OCPP Data. *Pages 85–94 of:* LEFER, W., & GRAVE, M. (eds), *Visualization in Scientific Computing '97*. Wien: Springer Verlag, Eurographics.

REINDERS, K.F.J., POST, F.H., & SPOELDER, H.J.W. 1999. Attribute-Based Feature Tracking. *In:* GRÖLLER, E., RIBARSKY, W., & LÖFFELMANN, H. (eds), *Data Visualization '99, Proc. Joint Eurographics IEEE TCCG Symposium on Visualization*. Wien: Springer Verlag.

RIVARA, M.-C. 1991. Local modification of meshes for adaptive and/or multigrid finite-element methods. *J. of Computational and Applied Mathematics*, **36**, 79–89.

ROBINSON, S.K. 1991. *The Kinematics of Turbulent Boundary Layer Structure*. NASA Technical Memorandum, no. 103859. Moffett Field, CA: NASA, Ames Research Center. Chap. 9: Vortices and their identification, pages 199–213.

ROTH, M., & PEIKERT, R. 1996. Flow Visualization for Turbomachinery Design. *Pages 381–384 of:* YAGEL, R., & NIELSON, G.M. (eds), *Proc. Visualization '96*. IEEE Computer Society Press.

SADARJOEN, I.A. 1994. *Algoritmen voor particle tracing in 3D kromlijnige roosters*. M.Sc. thesis, Delft University of Technology.

SADARJOEN, I.A., & POST, F.H. 1997 (4–8 September). Deformable Surface Techniques for Field Visualization. *Pages C109–C116 of:* FELLNER, D., & SZIRMAY-KALOS, L. (eds), *Proc. Eurographics '97*. Computer Graphics Forum, vol. 16(3).

SADARJOEN, I.A., & POST, F.H. 1999a. Geometric Techniques for Vortex Detection. *In:* GRÖLLER, E., RIBARSKY, W., & LÖFFELMANN, H. (eds), *Data Visualization '99, Proc. Joint Eurographics IEEE TCCG Symposium on Visualization*. Wien: Springer Verlag.

SADARJOEN, I.A., & POST, F.H. 1999b. Techniques and Applications of Deformable Surfaces. *In:* HAGEN, H., & RODRIAN, H.C. (eds), *To be published in: Proc. Dagstuhl'97 Seminar on Scientific Visualization*. Springer Verlag.

SADARJOEN, I.A., VAN WALSUM, T., HIN, A.J.S., & POST, F.H. 1994 (30 May – 1 June). Particle Tracing Algorithms for 3D Curvilinear Grids. *In: Proceedings of the Fifth Eurographics Workshop on Visualization in Scientific Computing*.

SADARJOEN, I.A., VAN WALSUM, T., HIN, A.J.S., & POST, F.H. 1997. Particle Tracing Algorithms for 3D Curvilinear Grids. *Chap. 14, pages 299–323 of:* NIELSON, G.M., HAGEN, H., & MÜLLER, H. (eds), *Scientific Visualization: Overviews, Methodologies, Techniques.* IEEE Computer Society Press.

SADARJOEN, I.A., DE BOER, A.J., POST, F.H., & MYNETT, A.E. 1998a. Particle Tracing in σ-Transformed Grids using Tetrahedral 6-Decomposition. *Pages 71–80 of:* BARTZ, D. (ed), *Visualization in Scientific Computing '98.* Wien: Springer Verlag, Eurographics.

SADARJOEN, I.A., POST, F.H., MA, B., BANKS, D.C., & PAGENDARM, H.G. 1998b. Selective Visualization of Vortices in Hydrodynamic Flows. *Pages 419–423 of:* EBERT, D., HAGEN, H., & RUSHMEIER, H. (eds), *Proc. Visualization '98.* ACM Press.

SERRA, J. 1982. *Image Analysis and Pattern Recognition.* London: Academic Press.

SHIH, C., & HO, C.M. 1994. 3D recirculation flow in a backward facing step. *ASME Journal of Fluids Engineering,* **116**(2), 228–232.

SILVER, D.S., POST, F.H., & SADARJOEN, I.A. 1999. Flow Visualization. *Chap. 7512 of:* WEBSTER, J.G. (ed), *Wiley Encyclopedia of Electrical and Electronics Engineering,* vol. Visualization and Computer Graphics. John Wiley & Sons, Inc.

STRID, T., RIZZI, A., & OPPELSTRUP, J. 1989. Development and Use of Some Flow Visualization Algorithms. *In: Computer Graphics and Flow Visualization in Computational Fluid Dynamics.* Lecture Series 1989-07. Von Kármán Institute for Fluid Dynamics.

TEITZEL, C., GROSSO, R., & ERTL, T. 1997. Efficient and Reliable Integration Methods for Particle Tracing in Unsteady Flows on Discrete Meshes. *Pages 31–41 of:* LEFER, W., & GRAVE, M. (eds), *Visualization in Scientific Computing '97.* Wien: Springer Verlag, Eurographics.

TRISULA USER GUIDE. 1993. *TRISULA. A simulation program for hydrodynamic flows and transports in 2 and 3 dimensions. User Guide Release 2.* WL | Delft Hydraulics.

TUFTE, E. 1990. *Envisioning Information.* Cheshire: Graphics Press.

TURK, G., & BANKS, D. 1996. Image-Guided Streamline Placement. *Pages 453–460 of:* RUSHMEIER, H. (ed), *Proc. SIGGRAPH 96.* Annual Conference Series. ACM SIGGRAPH.

VILLASENOR, J., & VINCENT, A. 1992. An algorithm for space recognition and time tracking of vorticity tubes in turbulence. *CVGIP: Image Understanding,* **55**, 27–35.

VAN WALSUM, T. 1995. *Selective Visualization Techniques for Curvilinear Grids.* Ph.D. thesis, Delft University of Technology, Delft.

VAN WALSUM, T., & POST, F.H. 1994. Selective visualization of vector fields. *Pages C339–C347 of:* DÆHLEN, M., & KJELLDAHL, L. (eds), *Eurographics '94*. Computer Graphics Forum, vol. 13(3). Oxford: Blackwell.

VAN WALSUM, T., POST, F.H., SILVER, D., & POST, F.J. 1996. Feature Extraction and Iconic Visualization. *IEEE Transactions on Visualization and Computer Graphics*, **2**(2), 111–119.

WILLIAMS, D.J., & SHAH, M. 1990. A Fast Algorithm for Active Contours. *Pages 592–595 of: Proceedings Third IEEE Intl. Conf. on Computer Vision*. International Journal of Computer Vision.

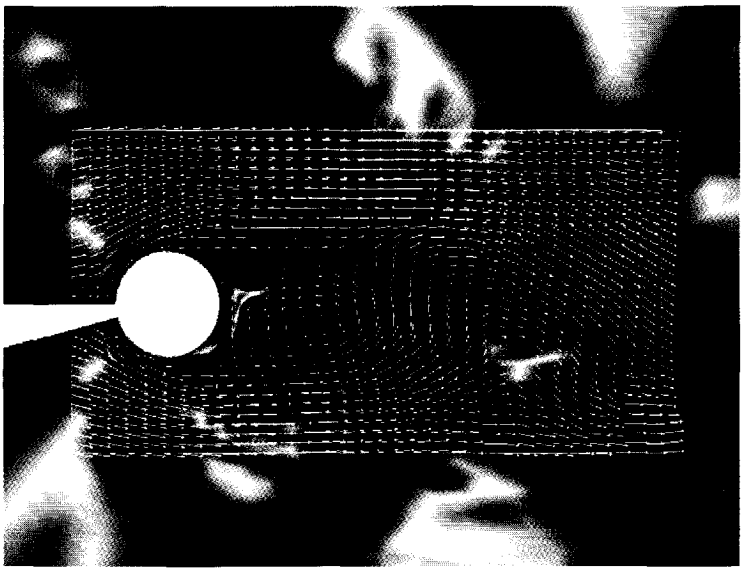
WITKIN, A.P., & HECKBERT, P.S. 1994. Using Particles to Sample and Control Implicit Surfaces. *Pages 269–277 of: Proc. SIGGRAPH '94*. The Association of Computing Machinery.

VAN DER WOUDEN, T. 1997. *Visualisatie met Spot Noise via Vis-a-Web en de CNX bibliotheek voor visualisatie met ongestructureerde roosters*. M.Sc. thesis, Delft University of Technology.

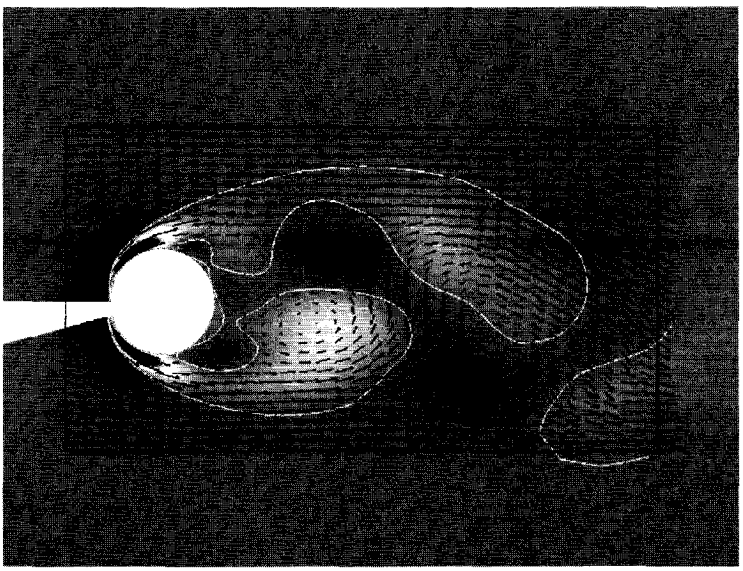
VAN WIJK, J.J. 1993. Implicit Stream Surfaces. *Pages 245–252 of:* NIELSON, G.M., & BERGERON, R.D. (eds), *Proceedings Visualization '93*. IEEE Computer Society Press, Los Alamitos, CA.

ZHU, Z.F., & MOORHEAD, R.J. 1995. Extracting and Visualizing Ocean Eddies in Time-Varying Flow Fields. *In: Proceedings of the 7th International Conference on Flow Visualization*.

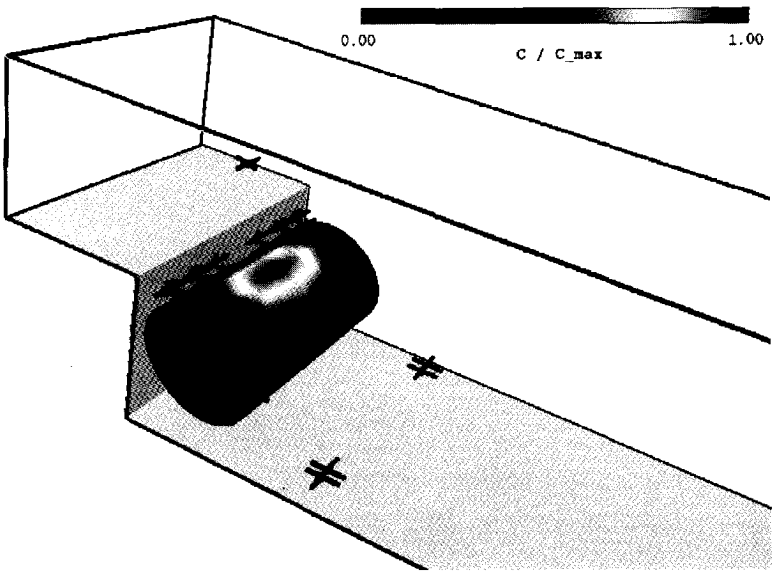
Colour Plates



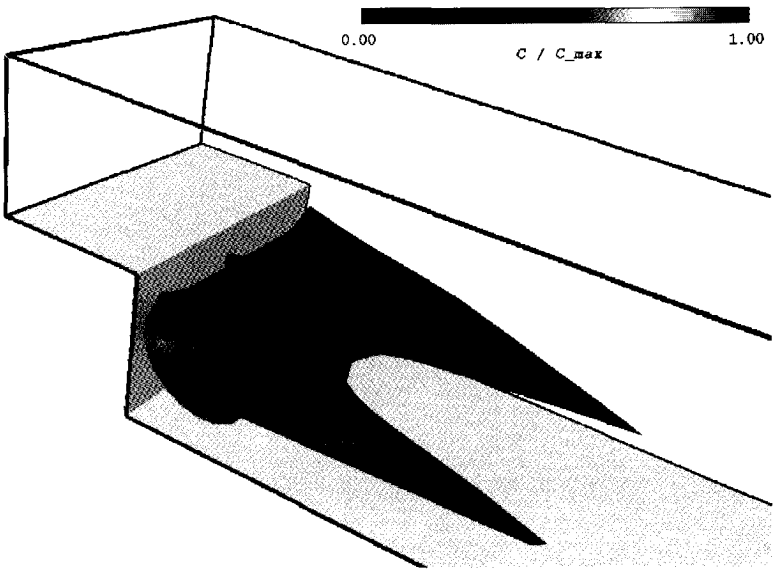
Colour Plate 1: Flow past a tapered cylinder, with velocity vectors and a slice coloured with normalized helicity. (See also Fig. 4.3a on p. 52.)



Colour Plate 2: Flow past a tapered cylinder, with velocity vectors and a slice coloured with λ_2. The white curves indicate $\lambda_2 = 0$. (See also Fig. 4.3b on p. 52.)



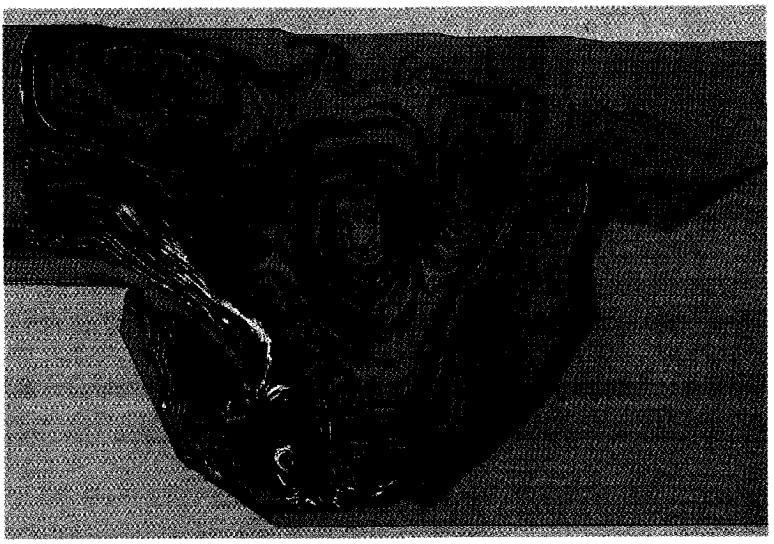
Colour Plate 3: Backward-facing step with selected nodes and an initial deformable surface coloured with the relative cost function. (See also Figs. 5.18–5.19 on pp. 89–90.)



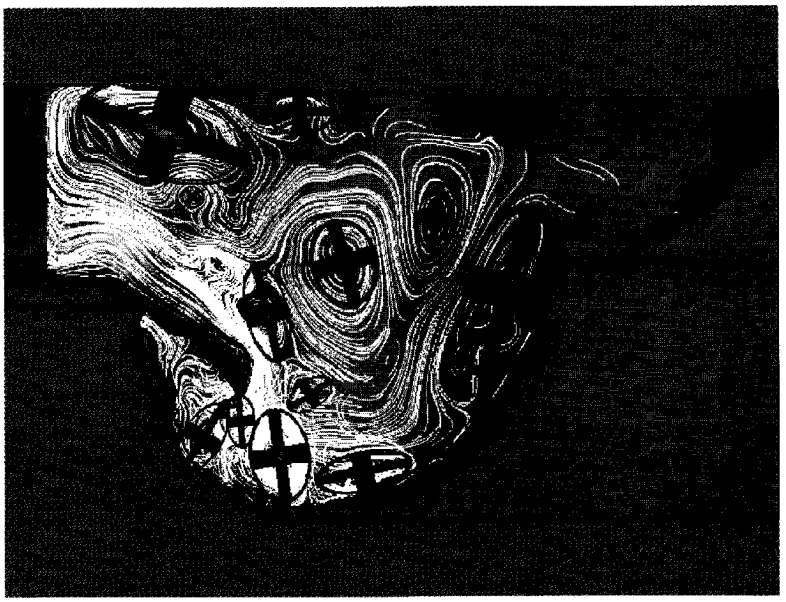
Colour Plate 4: Deformable surface approximating a recirculation zone. (See also Fig. 5.20 on p. 91.)

This page intentionally left blank

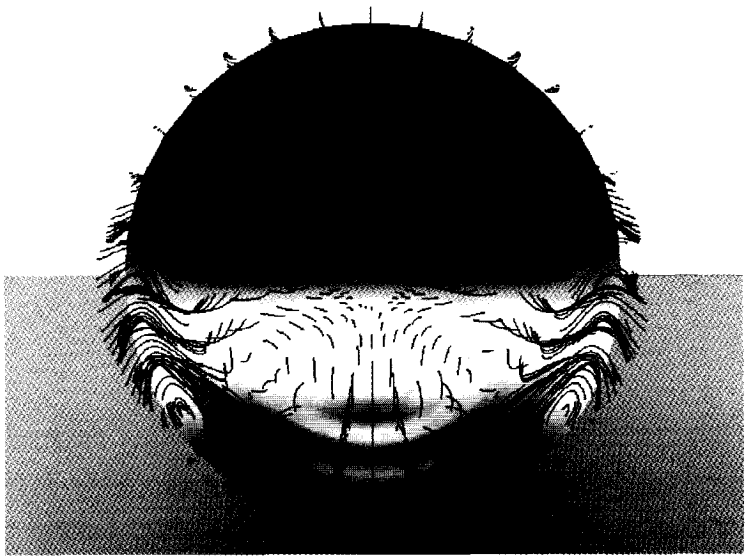
This page intentionally left blank



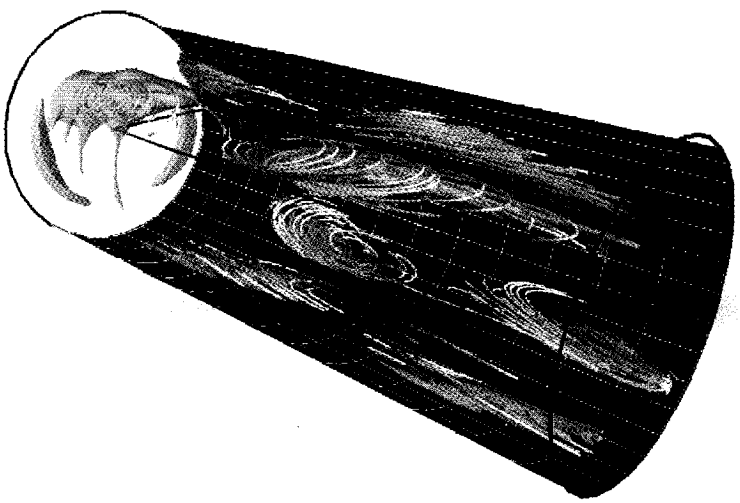
Colour Plate 5: Bay of Gdańsk with the global flow patterns visualized by streamlines in a horizontal grid slice. Colour indicates velocity magnitude. (See also Fig. 6.6 on p. 102.)



Colour Plate 6: Bay of Gdańsk with vortices approximated by ellipses. Red ellipses are clockwise, green ellipses counterclockwise vortices. (See also Fig. 6.13 on p. 110.)



Colour Plate 7: Pipe flow with particle paths and a slice coloured with pressure. (See also Fig. 6.16a on p. 113.)



Colour Plate 8: Pipe flow with semi-transparent isosurfaces approximating vortices and selective streamlines through the regions. (See also Fig. 6.21 on p. 117.)

Summary

This thesis describes the extraction and visualization of geometries in fluid flows, with the goal of gaining insight into important features of the flow. Geometries are curves, surfaces, and volumes, which have the advantage that they can be easily perceived by the human visual system, and efficiently rendered by computers.

Chapter 2 contains a survey of related work in the field of geometric techniques. This serves as a framework for placing the techniques covered in the following chapters into context.

Chapter 3 covers 'particle tracing', a technique which simulates massless particles transported by a flow. By calculating and visualizing the particle paths, information is obtained about the flow features. Complications are caused by the use of grids with irregularly formed cells, or with irregular topologies. In such grids, point location (determining which cell contains a certain particle) is a problem. This chapter describes particle tracing techniques for curvilinear grids, for so-called σ-transformed grids, and for unstructured grids. In curvilinear grids consisting of hexahedral cells, the problem can be solved by decomposing the cells into 5 tetrahedra. In σ-transformed grids, where the cells are often strongly deformed, this method is inadequate. This can be solved by decomposing the cells into 6 tetrahedra. In unstructured grids, the cells are often tetrahedra, but extra connectivity information must be added to solve the point location problem. Once this information has been derived, particle tracing in unstructured grids is well possible.

Chapter 4 covers techniques for detecting vortices, an important class of flow features. These techniques may employ two types of criteria: physical criteria and geometric criteria. The first category is based on physical quantities, such as pressure and vorticity. It has turned out that these criteria do not detect all vortices; especially weak vortices, characterized by low velocities, are often missed. The second category, which uses geometric criteria, is only based upon shape properties of flow curves, such as particle paths. It is assumed that vortices are characterized by rotational movement, which leads to circular particle paths. In this category, two methods have been developed and evaluated: one method is based on the curvature centres of curves, another on their winding-angles. It turns out that the curvature centre method is quite sensitive to not perfectly circular vortices, in which case it does not detect them. The winding-angle method often does find weak and non-circular vortices. In addition,

this method offers the capability of quantifying vortices by calculating numerical attributes of them.

Chapter 5 covers deformable surfaces. These are surfaces which start from a given initial shape, and are deformed to a target shape in a number of iteration steps. The target shape corresponds to a feature to be extracted from the flow. Deformation is accomplished by displacing points of the surface. Therefore, two important aspects of the deformation are determination of the step direction and of the step size. For determining the step direction, several alternatives have been investigated and compared. For determining the step size, a new method has been developed, which brings about a significant acceleration compared to existing methods. When a surface becomes too large and too coarse, it is necessary to refine it. Several methods for surface refinement described in the literature have been investigated and implemented. The chapter concludes with three applications of deformable surfaces: the extraction of local isosurfaces, recirculation zones, and vortices.

In Chapter 6, the techniques described in the preceding chapters are applied to four comprehensive cases: simulations of the Pacific Ocean and of the Bay of Gdańsk, of a pipe flow in transition between laminar and turbulent flow, and of the flow around a Delta Wing aircraft. In each case, several techniques are applied, and when several techniques are available for the same purpose, they are compared.

Samenvatting

Dit proefschrift beschrijft de extractie en visualisatie van geometrieën in vloeistofstromingen, met het doel inzicht te krijgen in belangrijke verschijnselen in de stroming. Geometrieën zijn krommen, oppervlakken en volumes, welke als voordeel hebben dat ze goed waar te nemen zijn door het menselijk waarnemingssyteem en efficiënt weer te geven zijn door computers.

Hoofdstuk 2 geeft een overzicht van verwant werk op het gebied van geometrische technieken voor visualisatie. Dit dient om de in de latere hoofdstukken behandelde technieken te kunnen plaatsen en te motiveren ten op zichte van verwant onderzoek.

Hoofdstuk 3 behandelt 'particle tracing', een techniek waarmee deeltjes gesimuleerd worden die zich door een stroming bewegen. Door de deeltjesbanen te berekenen en te visualiseren, wordt informatie verkregen over de verschijnselen in de stroming. Complicaties worden veroorzaakt door het gebruik van roosters met onregelmatig gevormde cellen of met een onregelmatige topologie. Hierdoor is de plaatsbepaling van deeltjes een probleem. Het hoofdstuk beschrijft particle tracing technieken voor kromlijnige roosters, voor zogenaamde σ-getransformeerde roosters en voor ongestructureerde roosters. Bij kromlijnige roosters die uit zesvlakkige cellen bestaan, kan het probleem opgelost worden door de cellen in 5 tetraëders op te delen. Bij σ-getransformeerde roosters, waarin cellen vaak sterk vervormd zijn, schiet deze methode tekort en is opdelen van de cellen in 6 tetraëders een oplossing. Bij ongestructureerde roosters zijn de cellen vaak al tetraëders, maar moet extra connectiviteitsinformatie toegevoegd worden om het plaatsbepalingsprobleem op te lossen. Wanneer deze informatie is bepaald, is particle tracing in ongestructureerde roosters goed mogelijk.

Hoofdstuk 4 behandelt technieken voor het detecteren van wervels, één van de belangrijkste stromingsverschijnselen. Deze technieken kunnen gebruik maken van twee soorten criteria: fysische criteria en geometrische criteria. De eerste categorie is gebaseerd op fysische grootheden, zoals druk en vorticiteit. Het is gebleken dat deze criteria lang niet alle wervels detecteren; vooral zwakke wervels, die gekenmerkt worden door lage snelheden, worden vaak gemist. De tweede categorie, die gebruik maakt van geometrische criteria, is uitsluitend gebaseerd op vormeigenschappen van stromingskrommen, zoals stroomlijnen of deeltjesbanen. Er wordt aangenomen dat wervels gekenmerkt worden door een rondgaande beweging, wat zich uit in lussen

in deeltjesbanen. In deze categorie zijn twee methoden ontwikkeld en geëvalueerd: één methode gebaseerd op krommingsmiddelpunten ('curvature centres') en één gebaseerd op windingshoeken ('winding-angles') van krommen. Het blijkt dat de curvature centre methode erg gevoelig is voor niet volmaakt cirkelvormige deeltjesbanen en in dergelijke gevallen geen wervels vindt. De winding-angle methode vindt zowel zwakke als niet-cirkelvormige wervels vaak wel. Bovendien biedt deze methode de mogelijkheid om wervels te quantificeren door numerieke attributen ervan te berekenen.

Hoofdstuk 5 behandelt deformeerbare oppervlakken. Dit zijn oppervlakken die vanuit een gegeven beginvorm in meerdere stappen vervormd worden tot een bepaalde doelvorm. In de regel begint een oppervlak klein en groeit het naar deze doelvorm toe. De doelvorm komt overeen met een te vinden verschijnsel in de stroming. Vervorming komt tot stand door punten van het oppervlak te verplaatsen, in een bepaalde richting en in een bepaalde afstand. Twee belangrijke aspecten van het vervormen van oppervlakken zijn daarom: het bepalen van de staprichting en van de stapgrootte. Voor het bepalen van de staprichting zijn verschillende alternatieven onderzocht en vergeleken. Voor het bepalen van de stapgrootte is een nieuwe methode ontwikkeld, die een aanzienlijke snelheidsverbetering met zich meebrengt ten opzichte van de gebruikelijke methoden. Wanneer een oppervlak te groot en te grof wordt, vanwege te grote driehoekjes, is het nodig om het te verfijnen. Voor het verfijnen van oppervlakken zijn enkele in de literatuur beschreven mogelijkheden onderzocht en geïmplementeerd. Het hoofdstuk sluit af met drie toepassingen van deformeerbare oppervlakken: het extraheren van locale iso-oppervlakken, recirculatiezones en wervels.

In hoofdstuk 6 worden de in de voorgaande hoofdstukken behandelde technieken toegepast op vier uitgebreidere gevallen: simulaties van de Stille Oceaan en de Baai van Gdańsk, van een pijpstroming die overgaat van laminair naar turbulent en van een stroming rond een Delta-vleugel-vliegtuig. Voor ieder geval worden verscheidene technieken toegepast en wanneer voor een bepaald doel meerdere technieken beschikbaar zijn, worden zij met elkaar vergeleken.

Curriculum Vitæ

Ari Sadarjoen was born on the 24^{th} of November 1968 in Mainz (Germany). In 1987, he received his VWO diploma from the Bonaventura College in Leiden (The Netherlands). In the same year, he started studying computer science at Delft University of Technology. In 1994, he received his MSc degree from the Computer Graphics / CAD-CAM group. The title of his Master's thesis was "Particle Tracing Algorithms for 3D Curvilinear Grids", part of which was presented at an international conference.

In September 1994, he started his PhD project in the same group, in a project in the field of scientific visualization. During his contract period, he also contributed to the teaching activities in the group.

In January 1999, he joined the Manchester Visualization Centre of The University of Manchester (U.K.), to conduct research in an EU project in the field of medical visualization. He can be reached through e-mail as `Ari.Sadarjoen@mcc.ac.uk`